DE NOVO PROTEIN DESIGN OF NOVEL FOLDS USING GUIDED CONDITIONAL WASSERSTEIN GENERATIVE ADVERSARIAL NETWORKS (GCWGAN)

A Thesis

by

SHAOWEN ZHU

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

August 2019

Major Subject: Electrical Engineering

ABSTRACT

In the research areas about proteins, it is always a significant topic to detect the sequence-structure-function relationship. Fundamental questions remain for this topic: How much could current data alone reveal deep insights about such relationship? And how much could such insights enable inverse protein design, the design of protein sequences for desired structures or functions? In this project two novel generative models, the conditional Wasserstein GAN (cWGAN) and guided conditional Wasserstein GAN (gcWGAN), are developed to generate new sequences for a structure fold that is desired and novel. We first mapped the fold space into a low-dimensional Euclidean space in order for the fold representation. We also used a fast fold prediction method as the oracle and a feedback in gcWGAN. To train our models, we used a semi-supervised learning process where both sequences with and without paired structures are exploited for model training. For the results we got, we analyzed the relationship between the model efficiency and factors such as the oracle's accuracy and data (sequence) availability; and we also found more diverse (and sometimes more novel) designs from gcWGAN compared to those from conditional VAE (variational autoencoder). These results reveal the value of current data in unraveling sequence-structure relationship and inverse protein design.

DEDICATION

This study is wholeheartedly dedicated to researchers who have been working on bioinformatic or machine learning areas.

ACKNOWLEDGMENTS

I would like to thank Dr. Yang Shen’ s group of the department of Electrical and Computer Engineering in Texas A&M University. Special thanks to Dr. Shen, Mostafa Karimi and Yue Cao, who are the collaborators of this project. I would also like to thank Dr. Jean-Francois Chamberland, Dr. Chao Tian and Dr. Sing-Hoi Sze for the review of this material. Also, we thank Texas A&M High Performance Research Computing for GPU allocations (https://hprc.tamu.edu).

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a thesis committee consisting of Professor Yang Shen (advisor), Professor Jean-Francois Chamberland and Professor Chao Tian of the Department of Electrical and Computer Engineering and Professor Sing-Hoi Sze of the Department of Computer Science and Engineering. This project was completed with my colleagues, Mostafa Karimi and Yue Cao. The data analyzed for this project was from SCOPe (v. 2.07) and UniRef50.

Funding Sources

This work was supported by the National Institutes of Health (R35GM124952).

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1 Sequence-structure-function Relationship

In the research areas about protein, it is always a fundamental theme to find out the relationship of sequence-structure-function [1]. In recent years, various bioinformatic topics based on such areas have been emerging in order to discover or make use of this relationship, such as structure prediction from sequence [2] and sequence design for desired structures [3, 4]. Particularly, in [5] Anfinsen and co-workers studied the renaturation of fully denatured ribonuclease and eventually established a thermodynamic hypothesis that native conformations of proteins in physiological milieu correspond to the lowest Gibbs free energy of the whole system [6]. After that there have been a significant progress in algorithmic development for the direct exploration on the sequence-structure relationship. This exploration is often pursued in two directions, the forward problem of protein structure prediction (also called protein folding sometimes) and the inverse problem of protein (sequence) design. In recent years, the data of protein structure and sequence keeps accumulating quickly, and then there comes another central question that based on current data to what we can extent through a deep insights about sequence-structure relationships, and whether we may design new proteins sequences for some known folds or even some novel folds just according to this relationship.

1.2 Forward Problem of Protein Structure Prediction

The forward problem of protein structure prediction often refers to the problem of protein folding, which means to predict the structure (or protein folds) according to a known sequence. One special example is the *ab initio* prediction that has often been solved by energy minimization without templates. However, with the increasing data size of protein structure and sequences, even this classical principle-driven approach has greatly benefited from the data, like the structural fragments has been applied for efficient sampling, and the sequence data has been used for training scoring functions. In recent years there comes a wave of data on protein sequences without paired structures. For instance, the residue-level sequence co-evolution in proteins [7] has been exploited for residue-residue structure contacts (contact map) using direct coupling analysis [8, 9, 10, 11], and the long-range residue contacts are especially useful to enhance protein structure prediction significantly [12, 11, 13]. In the latest CASP (Critical Assessment of Structure Prediction) rounds, there is a more recent revolution that is based on the prediction of residue-residue contact distance (or distance distribution) even for proteins with few homolog sequences, which is enabled by advanced deep neural network architectures (especially deep residual networks) that learn from the sequence, structure, and co-evolution data [14, 15].

1.3 Inverse Problem of Protein Design

The inverse problem of protein design is opposite to the forward problem that it often asks for designing some desired sequences given a protein fold or a structure. It is also often pursued following the energy minimum principle whereas the additional complication comes from the combinatory in the sequence space [16, 17, 18]. Currently there are three main classes of such design algorithms, the exact algorithms, the approximation algorithms and the heuristic algorithms. For the exact algorithms there are some recently developed methods like DEE, A^*, and cost function networks [19, 20, 21, 22]; for the approximation algorithms there are relaxed integer-programming and loopy belief propagation [23, 24]; and the genetic algorithms developed in 1994 and Markov chain Monte Carlo (MCMC) developed in 2011 [25, 26] belong to the heuristic algorithms. Some early methods such as specificity redesign [27, 28] and antibody potency enhancement [29] have also reached great success, and these methods could also be extended to *de novo* protein design where even the exact backbone structure is assumed unknown along with the sequence [30] using multiple backbone conformations from fragments or geometry. Particularly, the energy minimum principle-driven Rosetta tools [31, 32, 26] have explored uncharted sequence space for *de novo* protein design. There are also some recent success stories for them including proteins with curved $beta$-sheets accommodating variously shaped cavities [33], self-assembling helical protein filaments [34], a fluorescence-activating β-barrel [35], and potent and selective mimics of IL-2 and IL-15 [36].

However, the inverse problem of *de novo* protein design has not witnessed so significant impacts from deeply exploiting data with advanced artificial intelligence (especially deep learning) technologies, which is not similar to the forward problem of (*ab initio*) protein structure prediction. But at the same time, the impacts of deep generative models, represented by Generative adversarial Networks (GAN) [37] and Variational Auto-Encoder (VAE) [38], have reached the sibling fields of inverse design for DNA [39, 40], RNA [41], small molecules [42, 43], and peptides [44]. Therefore, there comes a main topic of this study that what the unique challenges for deep generative models can be in *de novo* protein design for a novel fold. These challenges can come from the design space, the attribute or property space (for design objective), and the mapping in between. The first challenge is a numerical one that comes from the much more daunting protein sequence space. Specifically speaking, protein sequences have more choices at each position compared to aforementioned molecular design problems (20 standard amino acids versus 4 nucleotides) and are much longer, leading to a dimensionality of $20^L \gg 4^K$ where $L > K$. The second challenge, a conceptual and mathematical one, is that the fold space is a discrete domain that has not been completely observed [45] while we also need a generalizable representation for the novel folds (values in the discrete space) that have never been seen in training data. In contrast, aforementioned deep generative small-molecule designs often target either a continuous property (such as logP) with a range of values or a discrete one with all possible values observed in training data. The last challenge comes from the knowledge gap in the sequence-fold relationship, as protein folds are products of both convergent and divergent evolution [46]. For example, very similar sequences often have the same fold with not-so-rare exceptions; and very dissimilar sequences can also have the same fold. But for designing RNAs, the problem often benefits from the fact that desired structures can often be readily translated to base pairing patterns in the sequence space.

1.4 An Introduction of Our Work

In this project we presented a study that is aiming at exploiting current data and advanced computing technologies for a faster, broader, and deeper exploration of the protein sequence space and seeking principles underlying protein structure folds at the same time. We have developed two deep networks based on some powerful algorithms. To overcome the aforementioned challenges, we have developed a semi-supervised guided conditional Wasserstein Generative Adversarial Network (gcWGAN) by exploiting both a generalizable low-dimensional global fold representation, protein sequence data without paired structures, and feedback from a fold oracle. Specifically speaking, we applied kernel Principal Correlation Analysis (kPCA) to get a fold represention over 1,232 known folds and to be generalizable for novel folds. We then develop a novel Generative Adversarial Network (GAN) for protein sequence generation to be conditioned on the low-dimensional fold representation and we called it conditional Wasserstein GAN. Based on that we also constructed a model that is guided through a protein fold prediction algorithm (oracle) named DeepSF, which is the gcWGAN model. For gcWGAN we pre-trained the highly complex deep generative model using unsupervised protein sequence data in order to increase the stability of the model. We then rigorously and systematically evaluate the performance of our models over a diverse fold space so as to generate valid and novel protein sequences for novel folds that are not in the training set. Moreover, to further check the generalizability of the models, we also test them on a recent novel fold that is not even in the original list of 1,232 folds.

2. MATERIALS AND METHODS

2.1 Data

2.1.1 Pre-processing and Classification

The fold data, including the folds in the structure domains and their corresponding protein sequences we used for this project is downloaded from SCOPe (V. 2.07), and we filtered the sequences at 100% identity level, which means there are not two sequences with a 100% identity (exactly the same) in our data set. Besides, for some sequences (<2%) there are some non-standard amino acid letters ('b', 'z', 'x' and 'X') inside them, and we regarded these sequences to be uncommon. For the non-standard cases, 'b', 'z' and 'x' respectively represent the ambiguity among ['d', 'n'], ['q', 'e'] and all 20 amino acids, and we assigned the explicit standard amino acids to every occasion of them based on a uniform prior distribution; 'X' represent a gap in the sequence, and we just disregarded these sequences for simplicity. Considering the limited budget on time and computational cost, we pursued this project and make the related analysis only on the sequences with a length between 60 and 160 (including 60 and 160) and disregarded the rest ones. The sequence length distribution is shown in Figure 2.1. As we constructed a convolutional neural network to be our generator and the length of its output (or the input sample of the critic) need to be fixed, we set that length to be the maximum of the interval, which is 160 in our project, and for the sequences shorter than that length we added padding at the end of them. Therefore considering balancing the range and the fold space coverage, we chose this interval. As a result, there are 781 of 1,232 total folds and over 35% of the sequences covered in our data. Besides, we also downloaded unpaired data, which means there are only protein sequences without the related structure, from UniRef50. We used these unpaired data for warming up the model.

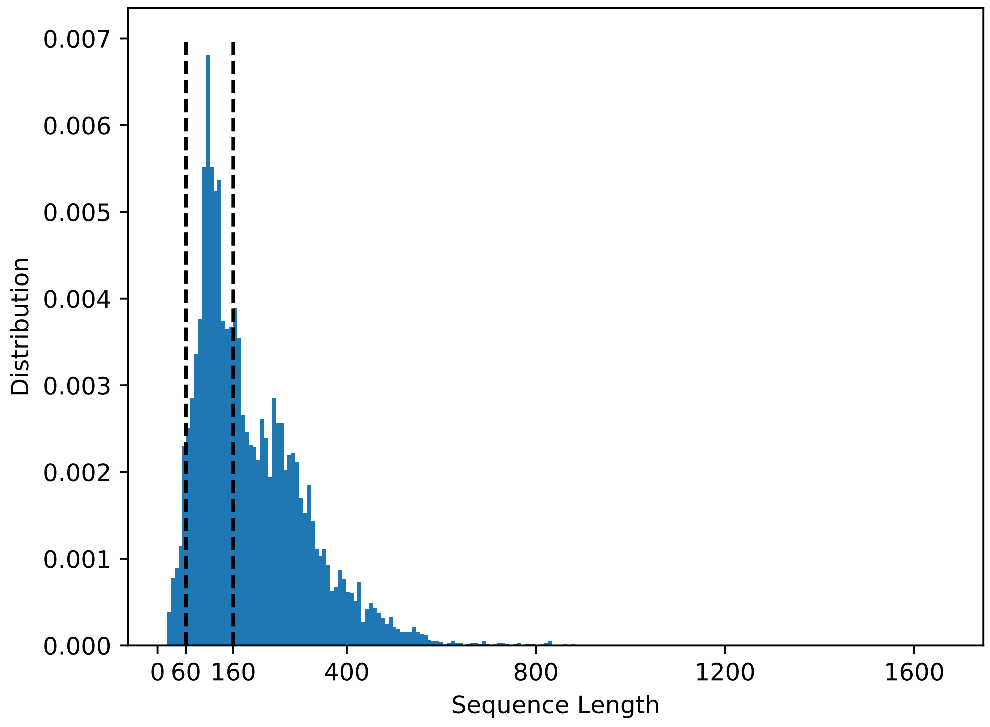


Figure 2.1: Sequence length distribution in the SCOPe data (100% sequence identity level). The percentage within the interval [60, 160] is 35.87%.

These sequences have also been clustered into 7 classes indexed from a to g in SCOPe. Besides, according to the sequence abundance (we also called it sequence availability or data availability) of each fold, we also classified the folds into three difficulty categories, easy, medium and hard, while easy refers to those whose sequence abundance is larger than 50, medium refers to those with 6 to 50 sequences and hard for the rest ones. The names for these classes and categories and the counts are shown in Table 2.1.

2.1.2 Data Splitting

We split our resulting dataset according to stratified sampling into training (70%), validation (15%) and test (15%) sets, while all of the three sets preserve the original fold-class distribution.

Class Index	Class Name	Easy	Medium	Hard	Total
a	α	27	64	105	196
b	β	28	57	54	139
c	α/β	15	35	18	68
d	$\alpha+\beta$	38	110	140	288
e	Multi-domain proteins	0	0	6	6
f	Membrane and cell surface	2	9	10	21
g	Small proteins	7	17	39	63
	Total	117	292	372	781

Table 2.1: The statistic of folds in the whole dataset based on class and difficulty.

The sequence statistics of the training set and the fold statistics of all are shown in Table 2.2 and Table 2.3 separately. As there is not overlapping between any two of the three sets, we can regard the folds in the validation set and test set as novel folds against those in the training set. For our two models, the hyper-parameters of the cWGAN were tuned only according to the critic loss, the nonsense sequence ratio, the padding ratio and the sequence identity while training, so both the validation and test sets played the same role for this model that we used them to estimate the performance; we used the validation set to tune one of the hyper-parameters of the guided-cWGAN model and the test set was used the same as before.

Class Index	Class Name	Training Set	Val. Set	Test Set	Total
a	α	139	29	28	196
b	β	109	15	15	139
c	α/β	52	8	8	68
d	$\alpha+\beta$	206	41	41	288
e	Multi-domain proteins	3	2	1	6
f	Membrane and cell surface	14	4	3	21
g	Small proteins	40	12	11	63
	Total	563	111	107	781

Table 2.2: The statistics of folds in training, validation and testing sets based on class.

Class Index	Class Name	# of Seq.
a	α	5486
b	β	4562
c	α/β	2536
d	$\alpha + \beta$	6625
e	Multi-domain proteins	8
f	Membrane and cell surface	189
g	Small proteins	719
	Total	20125

Table 2.3: The statistics of sequences in the training set.

2.1.3 Representative Sequences for Each Fold

One of the criteria for hyper-parameter tuning is the sequence novelty, for which we need to calculate the pairwise sequence identities between the generated sequences and the nature sequences related to the same fold. This process can be really time-consuming as we will discuss in section 2.6.1 . Therefore, for each fold in the training set, we would like to select several representative sequences and just calculate the sequence identities between the generated ones and them. For all the training folds, We firstly calculated all the pairwise sequence identities between each pair of sequences related to the same fold. As the sequence identity can be regarded a similarity measure, we then applied DBSCAN [47] algorithm and selected the sequences with the maximum sum of sequence identities with others in the same cluster to be the representative ones. Since the structure of the two sequences can be very similar if their sequence identity is larger than 0.3, we set ϵ to be 0.7 for the DBSCAN algorithm.

2.2 Fold Representation

The original data we got were distinct points in the fold structure space, where it can be hard to directly tell the relationship between the data and take it as the input of our model. Therefore we would like to use a low-dimensional fold representation that can preserve enough information of different folds, which means it can show the difference and relationships between different folds. Besides, we also hope the representation can be generalizable for describing the novel folds. In SCOPe we have got 1,232 basis folds, and recently the growth of the fold space seems to be near saturation [48], we just considered the space spanned by the 1,232 folds and applied kernel principal component analysis [49] to reduce the space dimension. Specifically, for the basis folds, we calculated the TM scores [50] between each two of them and constructed a pairwise similarity matrix. Then we added a properly -scaled identity matrix to it in order to get a positive-definite Gram matrix $\boldsymbol{S}$. It was then centralized through:

$$\begin{aligned} \boldsymbol{O}_{\text{NN}} &= \mathbf{1}_N \mathbf{1}_N^T / N; \\ \tilde{\boldsymbol{S}} &= (\boldsymbol{I} - \boldsymbol{O}_{\text{NN}}) \boldsymbol{S} (\boldsymbol{I} - \boldsymbol{O}_{\text{NN}}); \end{aligned} \tag{2.1}$$

where N is the row and column size of $\boldsymbol{S}$, which is 1,232 here, and $\mathbf{1}_N$ is an all-one vector of size N. After performing eigen-decomposition to the matrix $\tilde{\boldsymbol{S}}$ and ranking its eigenvalues u_i and its corresponding eigenvectors $\boldsymbol{v}_i$ in a decreasing order: $\{(u_1, \boldsymbol{v}_1), (u_2, \boldsymbol{v}_2), ..., (u_N, \boldsymbol{v}_N)\}$, we can then calculated the ith dimension basis of fold space through:

$$\boldsymbol{r}_i = \frac{\boldsymbol{v}_i}{\sqrt{u_i}} \tag{2.2}$$

For the novel ones, suppose we have M incoming new folds, we can firstly calculate their TM scores with the previous N basis folds and then construct a new matrix $\boldsymbol{S}^*$ with size $M \times N$, and then centralize it though

$$\begin{aligned} \boldsymbol{O}_{\text{MN}} &= \mathbf{1}_M \mathbf{1}_N^T / N; \\ \tilde{\boldsymbol{S}}^* &= \boldsymbol{S}^* - \boldsymbol{S}^* \boldsymbol{O}_{\text{NN}} - \boldsymbol{O}_{\text{MN}} \boldsymbol{S} + \boldsymbol{O}_{\text{MN}} \boldsymbol{S} \boldsymbol{O}_{\text{NN}}; \end{aligned} \tag{2.3}$$

Finally, according to the formula $\tilde{\boldsymbol{S}}^* \cdot \boldsymbol{r}_i$ we can get the coordinate vectors of these these M folds along the ith dimension.

2.3 Oracle for Protein Fold Prediction

To evaluate the accuracy of our models we need an oracle to tell the target fold of a known sequence. For our guided-WGAN model the oracle also plays a role in guiding sequence generation during the training process. Currently there does not exist a closed-form mathematical formula that can represent the function from protein sequences to the target folds, but there are some data-driven approaches that can be used for protein fold prediction, such as DeepSF [51]. DeepSF is a 1D deep convolutional neural network designed to classify protein sequences into 1,195 folds in their dataset. It can output a probability vector of 1,195 dimension through a softmax layer with each entry represent the probability for the sequence to be classified into the related fold. In our project we applied top 10 accuracy, which means if the real fold can be found in the 10 folds with the highest probability we regard this sequence to be a successful one.

For the original DeepSF, the input of the network consists of 4 parts, sequence represented by one-hot encoding, position-specific scoring matrix (PSSM), predicted secondary structure (SS) and predicted solvent accessibility (SA). Except the one-hot encoding sequence, all of the other three requires PSI-BLAST, which is computational expensive, for multiple sequence alignment, so the whole process for DeepSF prediction can cost lots of time and computing source and then it will be unfeasible to apply it for the guided-cWGAN with a huge number of sequence generated while training. Besides, the original model can only predict 1,195 classes which is less then the number of folds we used, so the error for it to be applied in our model can be larger and we cannot get any effective feedback from it for the missed folds. Therefore, we modified DeepSF with cosidering the SCOPe update and slightly increasing the folds number to 1,215, and just took the one-hot encoding sequence as the input alone. However, as we disgarded the three informative features, the model accuracy can be significantly impacted, so we also modified the architecture of DeepSF model as we constructed a wider and deeper network with residual blocks (The filter size was increased from 10 to 40, and the model was inserted 10 residual convolutional layers to be of 20 layers at last). There are two branches in the modified DeepSF and the filter sizes are 6 and 10 respectively so that we can capture short and long motif information and combine them together at the end of the network. There are blocks of convolution layer inside each branch with batch normalization and ReLU nonlinearity following. At the beginning and at the end of the model these blocks are non-residual, but in the middle they are in residual fashion which are connected through skip connections. A customized max pooling layer was implemented at the end of each branch to capture only the top m amino acids from the sequence, and it is required to deal with length varying sequences so that it will have fixed length of m (which is 30 in our model) at the output layer.

2.4 Conditional Wasserstein GAN

Generative Adversarial Network (GAN) [37] is a powerful class of generative models developed in recent years. GAN can be interpreted as a game between a generator G and a discriminator D, while G can generate artificial data that can be very close to real ones with a random noise to be the input and the goal of D is to distinguish the generated data from the real ones. In the past few years, GAN has been widely used in various areas, such as generating images of unprecedented quality of celebrities [52], image to image translation [53], text to image translation [54], creating anime characters [55], image inpainting [56], face aging [57] and music generation [58]. It has also been extended to a supervised generative model from an unsupervised one through conditional GAN [59] that generates images specific for each class. In this project, we also need to generate protein sequences according to a specified fold which can be regarded as the condition.

Though GANs are widely used, it can be notoriously hard for them to be trained due to some problems such as the difficulty to achieve Nash equilibrium [60], low dimensional support, vanishing gradient and mode collapsing [61]. To solve these problems, Wasserstein GAN (WGAN) [61, 62] was introduced, while it use the Wasserstein distance (or earth-mover) instead of KL-divergence, and replace the discriminator with notation critics. Therefore, we consider the conditional WGAN (cWGAN) (formulated with the Kantorovich-Rubinstein duality) with gradient penalty (a soft version of Lipschitz constraint modeled through the penalty on the norm of the gradient) to be the architecture of our first model. The formualtion is shown as below:

$$\begin{aligned}\min_G \max_D L_1 = & E_{\mathbf{x}\sim P_r}[D(\mathbf{x}|\mathbf{y}^{\text{embd}})] - E_{\tilde{\mathbf{x}}\sim P_g}[D(\tilde{\mathbf{x}}|\mathbf{y}^{\text{embd}})] \\ & - \lambda E_{\hat{\mathbf{x}}\sim P_{\hat{\mathbf{x}}}}[(||\nabla_{\hat{\mathbf{x}}} D(\hat{\mathbf{x}}|\mathbf{y}^{\text{embd}})||_2 - 1)^2]\end{aligned} \tag{2.4}$$

In the formula above $\mathbf{y}^{\text{embd}}$ represents the protein fold embedding explained in Section 2.2. $\mathbf{x}$ represents the real sequences generated from the real data distribution P_r, and $\tilde{\mathbf{x}}$ refers to the artificial sequences generated from the model distribution P_g, implicitly defined by noise distribution $\mathbf{z} \sim p(\mathbf{z})$ and $\tilde{\mathbf{x}} \sim G(\mathbf{z}|\mathbf{y}^{\text{embd}})$. Moreover, inspired by the fact that the optimal critic contains straight lines and l2 norm of the gradient is equal to 1, connecting coupled points from P_g and P_r, $\hat{\mathbf{x}} \sim P_{\hat{\mathbf{x}}}$ is implicitly defined as sampling uniformly along straight lines between pairs of points sampled from P_g and P_r for a given label $\mathbf{y}^{\text{embd}}$. λ is a hyper-parameter which can adjust the importance of the gradient penalty in the loss function. Pseudo-code of conditional WGAN with gradient penalty is shown in algorithm 1 below:

Algorithm 1 Conditional WGAN with gradient penalty

```
1:  Randomly initialize the parameter from scratch
2:  while $\theta$ has not converged do
3:      for t=1$\cdots$,$n_{critic}$ do
4:          Sample real data $\{\mathbf{x}^{(i)}, \mathbf{y}^{embd,(i)}\}_{i=1}^m \sim P_r$
5:          Sample noise $\{\mathbf{z}^{(i)}\}_{i=1}^m \sim p(\mathbf{z})$
6:          Sample random number $\{\epsilon^{(i)}\}_{i=1}^m \sim U[0,1]$
7:          $\{\tilde{\mathbf{x}}^{(i)}\}_{i=1}^m \leftarrow \{G_\theta(\mathbf{z}^{(i)}|\mathbf{y}^{embd,(i)})\}_{i=1}^m$
8:          $\{\hat{\mathbf{x}}^{(i)}\}_{i=1}^m \leftarrow \{\epsilon^{(i)}\mathbf{x}^{(i)} + (1-\epsilon^{(i)})\tilde{\mathbf{x}}^{(i)}\}_{i=1}^m$
9:          $L \leftarrow \frac{1}{m}\sum_{i=1}^m D_w(\mathbf{x}^{(i)}|\mathbf{y}^{embd,(i)}) - D_w(\tilde{\mathbf{x}}^{(i)}|\mathbf{y}^{embd,(i)}) - \lambda(||\nabla_{\hat{\mathbf{x}}^{(i)}} D_w(\hat{\mathbf{x}}^{(i)}|\mathbf{y}^{embd,(i)})||_2 - 1)^2$
10:         $w \leftarrow \text{Adam}(\nabla_w L, w, \alpha, \beta_1, \beta_2)$
11:     Sample a batch of noises $\{\mathbf{z}^{(i)}\}_{i=1}^m \sim p(\mathbf{z})$
12:     $\theta \leftarrow \text{Adam}(\nabla_\theta \frac{1}{m}\sum_{i=1}^m -D_w(G_\theta(\mathbf{z}^{(i)}|\mathbf{y}^{embd,(i)})), \theta, \alpha, \beta_1, \beta_2)$
```

We have developed the architecture of our model inspired by WGAN-GP [62]. In our model, both the generator and the discriminator are constructed based on residual blocks that contain 2 layers of RELU for nonlinearity and 1D convoilutional layers with the skip connection between the input and the output. We concatenated the fold representation with a random noise to be the input of the generator, and set the length of output L to be 160, which is also the maximal length of the generated sequence. The input is then mapped to a higher-dimensional space ($L \times 500$) through a linear mapping layer. There are 20 layers in total and we set 5 layers of residual blocks in order that the model can be more expressive and the gradient vanishing from connection skipping can be avoided. At the end, we used a 1D convolutional layer to reduce the dimension from $L \times 500$ to $L \times 21$ (20 characters representing the 20 amino acids and one for the padding) and a softmax layer to get the probability distribution for the 21 characters for each position. By choosing the character with the maximum probability for each position we can finally get a generated protein sequence.

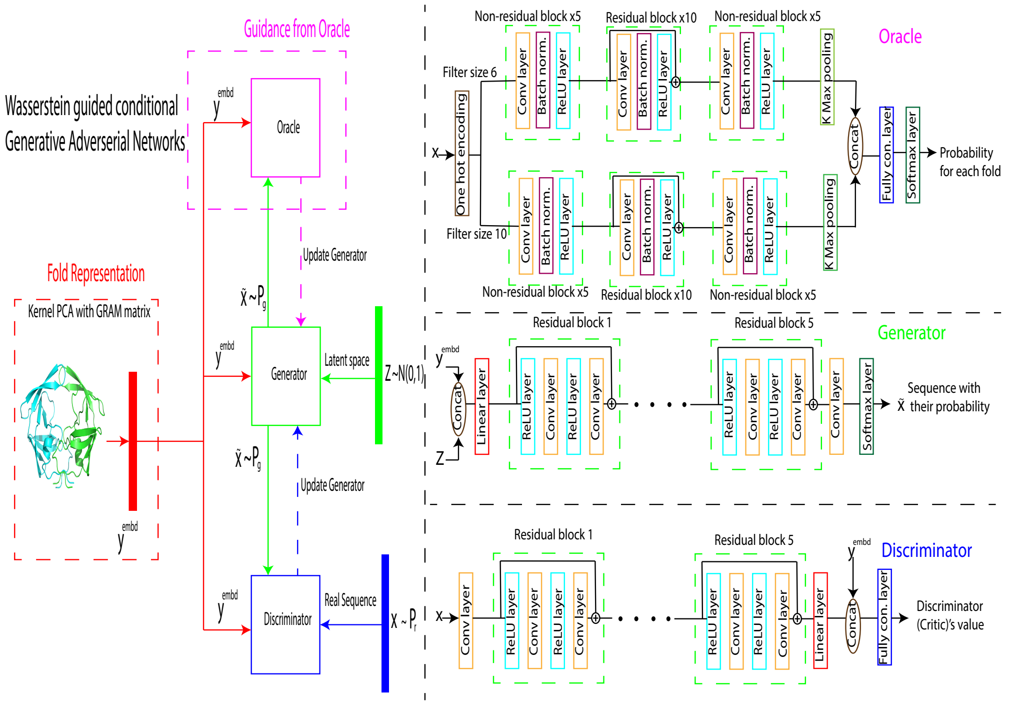


Figure 2.2: The architecture of out guided conditional Wasserstein GAN (gcWGAN).

2.5 Guided Conditional Wasserstein GAN

In principle, cWGAN tries to generate protein sequences given an specific desired fold, but then there can be a practical barrier due to the limitation of feature embedding for fold representation. To improve the chance of generating relevant protein sequences, we then developed a novel GAN model which can exploit the feedback from the oracle and we called it guided conditional wasserstein GAN (guided cWGAN or gcWGAN).

Firstly, there remains a fundamental question that whether the generated sequences can be “protein-like", which whether the generated sequences can fold into stable and functional protein structures. During the training process for cWGAN, we also found some generated sequences can be apparently nonsense. However, there is not bioinformatics tools that can quantify or predict this property without predicting the structures, so we decided to provide a warm start for the gcWGAN model through semi-supervised learning. Specifically speaking, we exploited a large quantity of protein sequences without paired structure data from UniRef50 and fixed their $\mathbf{y}^{embd}$ at the center of all fold representations. Then we trained the cWGAN model based on these unsupervised data and fixed the input of the condition to be the mean of the previous conditions. To reduce the computation cost, for the hyper-parameters that has been tuned before, we just fixed them as the best ones we got while training cWGAN. After that, we re-train the cWGAN model using the previous supervised sequences with corresponding fold representations based on this warm start.

Both our two models share the same structure, and the architecture of the gcWGAN is shown in Figure 2.2. But for gcWGAN we changed the loss function by adding a feedback part from the prediction of the oracle, so this process can be regarded as a closed loop version of cWGAN model. The new term of the loss function is shown below:

$$R = -E_{\tilde{\mathbf{x}}\sim P_g}[y^{\mathrm{ds}} \log O(\tilde{\mathbf{x}}) + (1 - y^{\mathrm{ds}}) \log (1 - O(\tilde{\mathbf{x}}))] \tag{2.5}$$

where $O(\tilde{\mathbf{x}}) = \sum_{k=1}^{K} p_k$, and p_k is the probability corresponding to the k^{th} top prediction from the oracle (modified DeepSF). Besides, if the desired fold "y" is in the top K (which is 10 in our project) predictions, then y^{ds} is one, and it is zero in the other case. Then there comes another problem that this loss function is nondifferentiable cause we need to figure out the binary value of y^{ds}, and it is important to have proper gradient for the backpropagation because we applied the adaptive gradient descent based algorithm for the optimization. As a result, we made an approximation that $y^{\text{ds}} \approx ReLU(p_y - p_K)$ and $1 - y^{\text{ds}} \approx ReLU(p_K - p_y)$, where p_y refers to the probability corresponding to desired fold "y" from oracle's prediction. Based on that, as long as the desired fold "y" is inside the top K predictions, $ReLU(p_y - p_K)$ will be a continuous positive value and $ReLU(p_K - p_y)$ will be zero, and it will be vice a versa if the desired fold "y" is not inside the top K predictions. This introduced “regularization" term can be regarded as the cross entropy between approximated y^{ds} and the top K prediction from DeepSF. We then added it to the total loss function with a weight λ_2, which is a hyper-parameter that can balance the original loss function L_1 and the new regularization term R:

$$\min_{G} \max_{D} L_2 = L_1 + \lambda_2 R \tag{2.6}$$

During the training process of guided-cWGAN, the oracle has already been trained as explained in Section 2.3 and is fixed. Because of the warm start, both the generator and the critic are stable even at the of the training process. Therefore the GAN model is less likely to be unstable with the new loss function. Besides, as the model is more likely to generate “valid" protein sequences, the prediction from DeepSF can be more reliable since it is a data-driven deep learning model (Previously the model can generate lots of sequences with padding between amino acid characters at the beginning of the training process, and we called these sequence invalid ones.). Moreover, we applied the continuous distribution on each amino acid generated from the generator from the softmax layer as the approximation for the one-hot encoding so as to avoid the gradient differentiability challenges.

Pseudo-code of guided cWGAN with gradient penalty is shown in algorithm 2:

Algorithm 2 guided cWGAN with gradient penalty

```
1: Initialize the parameter from the previously trained semi-supervised cWGAN
2: while $\theta$ has not converged do
3:     for t=1$\cdots$,$n_{critic}$ do
4:         Sample real data $\{\mathbf{x}^{(i)}, \mathbf{y}^{embd,(i)}\}_{i=1}^{m} \sim P_r$
5:         Sample noise $\{\mathbf{z}^{(i)}\}_{i=1}^{m} \sim p(\mathbf{z})$
6:         Sample random number $\{\epsilon^{(i)}\}_{i=1}^{m} \sim U[0,1]$
7:         $\{\tilde{\mathbf{x}}^{(i)}\}_{i=1}^{m} \leftarrow \{G_\theta(\mathbf{z}^{(i)}|\mathbf{y}^{embd,(i)})\}_{i=1}^{m}$
8:         $\{\hat{\mathbf{x}}^{(i)}\}_{i=1}^{m} \leftarrow \{\epsilon^{(i)}\mathbf{x}^{(i)} + (1-\epsilon^{(i)})\tilde{\mathbf{x}}^{(i)}\}_{i=1}^{m}$
9:         $L_D \leftarrow \frac{1}{m}\sum_{i=1}^{m} D_w(\mathbf{x}^{(i)}|\mathbf{y}^{embd,(i)}) - D_w(\tilde{\mathbf{x}}^{(i)}|\mathbf{y}^{embd,(i)}) - \lambda_1(||\nabla_{\hat{\mathbf{x}}^{(i)}} D_w(\hat{\mathbf{x}}^{(i)}|\mathbf{y}^{embd,(i)})||_2 - 1)^2$
10:        $w \leftarrow \text{Adam}(\nabla_w L_D, w, \alpha, \beta_1, \beta_2)$
11:    Sample a batch of noises $\{\mathbf{z}^{(i)}\}_{i=1}^{m} \sim p(\mathbf{z})$
12:    Sample real data $\{\mathbf{x}^{(i)}, \mathbf{y}^{embd,(i)}\}_{i=1}^{m} \sim P_r$
13:    $L_G \leftarrow \frac{1}{m}\sum_{i=1}^{m} -D_w(G_\theta(\mathbf{z}^{(i)}|\mathbf{y}^{embd,(i)})) + \lambda_2[y^{ds,(i)} \log O(G_\theta(\mathbf{z}^{(i)}|\mathbf{y}^{embd,(i)})) + (1 - y^{\text{ds}}) \log(1 - O(G_\theta(\mathbf{z}^{(i)}|\mathbf{y}^{embd,(i)})))]$
14:    $\theta \leftarrow \text{Adam}(\nabla_\theta L_G, \theta, \alpha, \beta_1, \beta_2)$
```

2.6 Model Performance Evaluation

2.6.1 Hyper-parameter Tuning

Both checking the accuracy and tuning the hyper-parameter of the model require model performance evaluation. There are three hyper-parameters for the cWGAN model, the initial learning rate, the critic iteration number and the noise length; for the gcWGAN model there can be a hyper-parameter more which is the weight of the feedback penalty in the loss function, λ_2 that is described in Section 2.5. In the training process we applied Adam optimization algorithm, where the learning rate will keep changing during training, but the initial learning rate can still influence the final result, so we considered it as a hyper-parameter. For each training iteration, we would update the parameters for the generator once and that for the critic several times, and we called this number the critic iteration number. The critic iteration number is also a hyper-parameter as the critic cannot tell the difference between generated data and real ones if it is too small, but it can also be overfitting on the current generator if that hyper-parameter is too large. The noise length is simply the dimension of the input noise. Due to the limit source of computation, for each hyper-parameter we selected several candidates in a certain range and tuned these hyper-parameters sequentially, which means we did not test all the combinations of them but only changed one of them and fixed the others to be the original values in [62] unless they have been tuned, and then for this hyper-parameter we selected the one with the best performance and fixed it in the future. The order of the hyper-parameters to be tuned is the initial learning rate, the critic iteration number and the noise length. For the gcWGAN model we just fixed the hyper-paremeters that had been tuned while training the cWGAN model and then tuned the weight at the feedback penalty based on it.

For hyper-parameter tuning we selected four criteria to find out the optimal hyper-parameter combination, the convergence of the cWGAN, the “nonsense" sequence ratio, the padding ratio and the sequence novelty. For the gcWGAN model we also took the feedback penalty in the loss function as a criteria to tune the hyper-parameters. We estimated these criteria for every epoch and plotted the results to select the ones with the best performance. The convergence of the cWGAN was assessed based on the loss of the critic which is the lower the better, and we would like to find out whether it can converge mathematically at the end. The “nonsense" sequence ratio refers to the ratio of the “nonsense" sequences in all the generated sequences, where “nonsense" means there are some padding at the beginning of the sequence or between the residue characters as we had not seen these kinds sequences in the natural dataset before. The padding ratio refers to the ratio of the padding characters in all the generated characters. This criteria is not so important cause we can generate sequences with different length, but we also regarded it as a reference to reflect the stability of the model. As the goal of this project is not only generating valid sequences that can lead to a desired fold, but also discovering some novel sequences that have not been seen before, we also want that the sequences generated can keep a high sequence novelty. We calculated the sequence novelty by calculating the pairwise sequence identities between the generated sequences and the representative folds that related to the same fold and took an average for each epoch. The feedback penalty is described in Section 2.5, and it is a feedback from the oracle that can tell the accuracy of the model.

2.6.2 Yield Ratio (Discover Rate) Calculation

The yield ratio is defined as the ratio of the successful sequences in all the generated sequences, which is the main criteria that we used for evaluating our model. For each fold we generated several sequences based on our models and took them as the input of DeepSF. For each sequence we selected the top ten predictions with the highest probability from DeepSF and checked whether the original target fold can be found in them, and if so we then regarded this sequence to be a successful one. Considering the yield ratio for some folds can be rather low and we are likely to only get a yield ratio to be zero if we do not generate enough sequences, but the computing resource is limited, we applied a generating loop for each fold: Firstly for the certain fold we at least generated 1,000 valid folds (which means we have discarded the nonsense ones) and sent them to the oracle; then we checked whether the number of the successful sequences was less than 10, and we terminated the loop if we had already got at least ten successful ones, otherwise kept generating sequences until we get 10 successful ones; since we had hundreds of folds to be test, we also terminated the loop if we have already generated more than 100,000 sequences due to the budget of the computational cost.

2.7 Funneling Filters for Generated Sequences

Based our trained gcWGAN model, we sequentially added it with with some designed filters so that we can increase the accuracy of the final output without or with only a little sacrificing the sequence diversity and sequence novelty, and then constructed a more complete system to generate sequences with higher quality. The filters are described as below:

1. Nonsense sequence filter: Both for cWGAN and gcWGAN, the original generator can generate some nonsense sequences with padding at the beginning or middle of the sequences. The first filter is to discard these nonsense sequences and just remain the rest valid ones. This filter had already been applied in calculating the sequence novelty and the yield ratio.

2. Oracle (Modified DeepSF) filter: We then sent the generated valid sequences to the modified

DeepSF to check whether the target fold is among the top 10 predictions. This filter is described in the yield ratio test part, and we just kept the successful sequences and discarded the others. For a novel fold that is not in the training set of our oracle, we could not get a prediction to be the target fold, so we applied another criteria that we checked whether the “neighbor" folds are among the top 10 prediction. Specifically speaking, for the novel fold we firstly calculated its 20 dimensional representation and then calculated the $l2$ norm between it and the representations of other folds in the oracle data set. If the $l2$ norm was less than a threshold (0.3 in our project) we called the related fold in the dataset a “neighbor" fold, and if one of the “neighbor" folds was among the top 10 prediction, we regarded this sequence to be a successful one.

3. RaptorX filter: For the successful sequences we tranfered them into FASTA format and sent them to a webserver named RaptorX, which can predict the actual protein structure according to the input sequence. For each seqeuence the RaptorX would output five structures, and we calculated the TM-scores between them and the ground truth. We selected the one with the highest TM-score to be the prediction as it was likely to be the most similar to the original structure. Considering the computational burden, in this project we only applied this step on 1,000 sequences generated for a novel fold that had passed the above two filters. As we got 1,000 predictions, we applied our fold representation method on them and calculated the $l2$ norm between their representations and that of the ground truth. We then selected the 100 structures with the lowest $l2$ norm and applied K-means algorithm in the representation space to get 10 clusters. For each cluster we chose the one with the lowest $l2$ norm and finally got 10 sequences for the next step.

4. Rosetta filter: Rosetta *ab initio* is a protein structure prediction software which is very powerful but also computational demanding. Therefore we can only selected 10 sequences from the above steps due to the computational burden. For each of these 10 sequences, we applied Rosetta software and finally got their protein structure *ab initio* prediction for 10,000 tra-

jectories, which means we got 10,000 structure predictions for a filter. For each prediction we can calculate its coordinate according to our fold representation method, and then we can measure the $l2$ distance between it and that of the real structure. By selecting the predictions with the lowest $l2$ distance or applying clustering algorithms, we can then further select several representative structure predictions for some further analysis.

3. EXPERIMENTS AND RESULTS

3.1 Fold Reresentation

As we dicussed in Section 2.2, for the fold space spanned by the 1,232 basis folds we applied kernel PCA and got the cumulative distribution of the variance along each dimension, which is shown in Figure 3.1. Since we would like to apply a low-dimensional fold representation to reduce the difficulty for training, and from the figure we can see that 17.3% of the variance can be explained by just the first 20 principal components, so we chose the 20-dimensional space to be the representation of the fold space. We then applied K-means clustering algorithm on these 1,232 folds in the 1,232 dimension space to get n clusters, and then we can get a subspace formed by the centeriod of the n clusters. As we projected the first 20 principal components onto this subspace while varying n from 20 to 600, and calculated the explained variance, we got the result shown in Figure 3.2 and we can see that the explained variance of these 20 principal componenets reached 63% when $n = 100$.

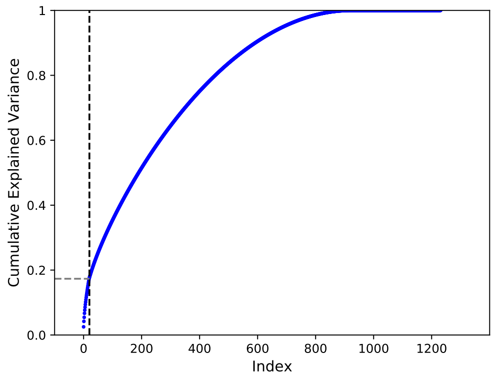


Figure 3.1: Top 20 eigenvalues of kernel PCA for fold representation.

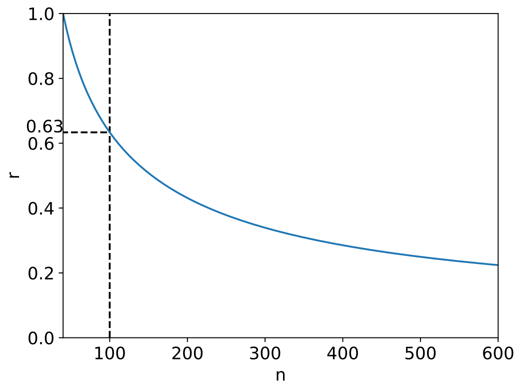


Figure 3.2: The ratio between the variance along the 20 projected eigenvectors and the total variance vs the number of clusters. When the number of clusters are chosen to be 100, the top 20 eigenvectors capture 63% variance.

Based on previous fold representation method we calculated the coordinates of the 1,232 folds and visualized it in a 3-dimensional space with the top 3 dimension (Figure 3.3). From the figure we can see that the folds were well-clustered according to different fold classes, and the folds belonging to class $\alpha\beta$ and class $\alpha + \beta$ distributed between the clusters of class α and class β, which can also satisfy our intuition.

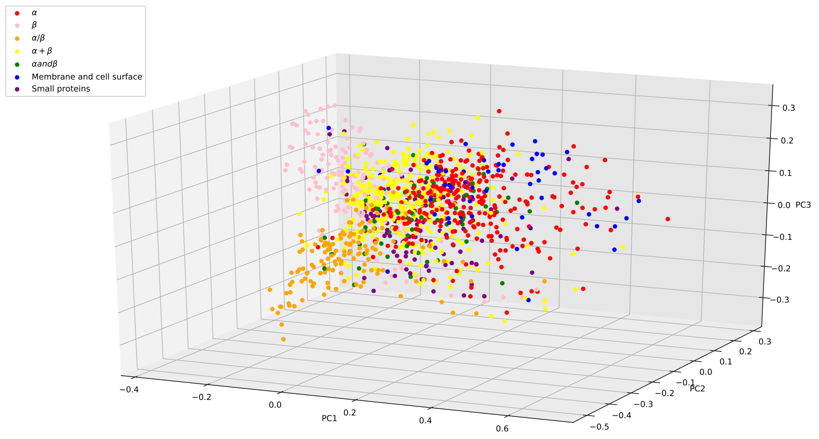


Figure 3.3: Visualization of 1,232 folds in a space spanned by the top 3 principal components found by kernel PCA.

As we hoped that our fold representation can reflect the relationship between the folds, we would like to see that after applying our method the folds can still keep the relationship with others, which means the distance between two similar folds can be still small but large for dissimilar ones. We can regard the TM-score as the similarity between two folds cause the larger it is, the more similar the two folds are. In our folds representation we represent each fold with a 20 dimensional vector, so we can use the $l2$ norm to reflect the distance between two folds. Figure 3.4 shows the relationship between the TM-scores and the $l2$ norms for same pairs of folds, and we can see that as the $l2$ norm gets larger, which means the distance between the two folds becomes larger, the similarity also reduces as the TM-score becomes smaller. The correlation is -0.4459. Therefore, our folds representation can finally reflect the relationship between different folds, and then it can be generalizable for novel folds.

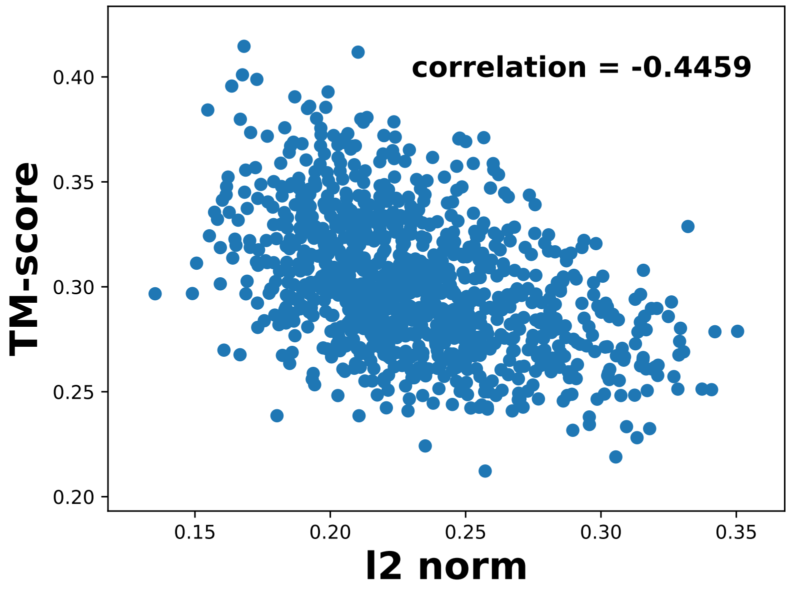


Figure 3.4: Relationship between l2 norm and TM-score.

3.2 Oracle's Accuracy

The oracle is the basic criteria to evaluate the accuracy of our model and our gcWGAN model is also guided by it, so its accuracy can strongly influence the accuracy of our generator. The original DeepSF can reach a high accuracy, but as we discussed in Section 2.3, it can be rather time-consuming to calculate the input features and its original data set is smaller than ours. Therefore we tried to modify the DeepSF model and got 3 new ones, and then we assessed and compared the performance of all of the 4 models: 1) the original DeepSF model taking all of the 4 features (one-hot encoding sequence feature + PSSM + SA + SS) as the input; 2) the original DeepSF model taking only one-hot encoding sequence features as the input; 3) a modified DeepSF model with only one-hot encoding sequence features; and 4) aforementioned modified DeepSF model with only one-hot encoding sequence features but using some more samples based on the updates of the recent SCOPe. The results we got are shown in Table 3.1, and we can see that the top 1 and top 10 prediction accuracy are 0.72% and 0.94% respectively for the the original DeepSF model with all 4 features. But the two kinds of accuracy drastically decreased to 0.33% and 0.69% respectively if we only took the one-hot encoding sequence features as the input. Theoretically speaking, the one-hot encoding sequence feature contains all the information cause we can calculated the other 3 features based on it, but it may be much more difficult to extract all the information directly from it and may ask for a more powerful network. As a result we modified the original model into a more deeper and complex neural network, and then we can increase the top 1 and top 10 prediction accuracy by 7% and 5%. To make the oracle fit our date set, we increased the dataset from 1,195 to 1,215 and retrained the model. The top 1 and top 10 prediction accuracy then decreased by 4% and 2% respectively. Finally, we decided to use the last ambiguous but fast model as our oracle, and the top 10 prediction with an accuracy of 72% as the criteria, which means we regarded a sequence to be successful to be successful it the target fold is among the top 10 predictions.

Prediction	Train set				Test set			
	All features	AA	AA+our	our+all fold	All features	AA	AA+our	our+all fold
Top1	0.74	0.36	0.56	0.49	0.72	0.33	0.40	0.36
Top5	0.93	0.65	0.82	0.75	0.9	0.59	0.63	0.61
Top10	0.97	0.76	0.89	0.84	0.94	0.69	0.74	0.72
Top15	0.98	0.82	0.92	0.88	0.96	0.74	0.79	0.78
Top20	0.98	0.86	0.94	0.91	0.97	0.79	0.82	0.81

Table 3.1: Original and modified DeepSF models' accuracy for protein fold prediction.

3.3 Hyper-parameter Tuning

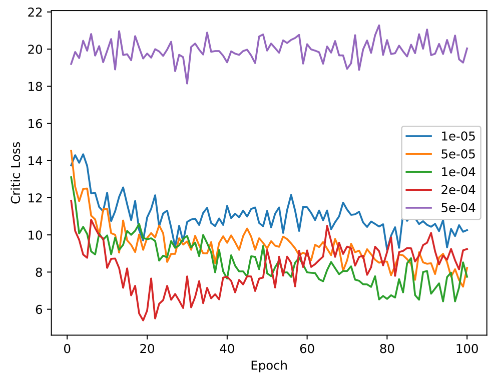
As we described in Section 2.6.1, there are 3 hyper-parameters to be tuned (initial learning rate, the critic iteration number and the noise length) for the cWGAN model and one more hyper-parameter (the weight of the feedback penalty in the loss function) for the gcWGAN model. In [62] the original value of the three hyper-parameters were 0.0001, 10 and 128 respectively, and we then tuned them in order according to the four criteria mentioned in Section 2.6.1. For the gcWGAN we fixed the three values to be the best ones we got from cWGAN and only tuned the feedback penalty weight according to to the feedback penalty loss on the validation set.

3.3.1 Initial Learning Rate

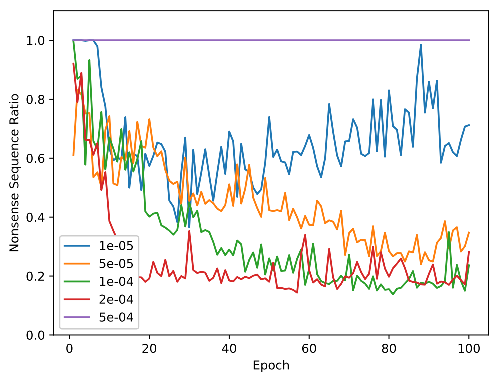
The first hyper-parameter we tuned was the initial learning rate for the Adam Optimizer which we applied to optimize the loss function during the training process, and we have talked about this hyper-parameter in Section 2.6.1. For the cWGAN model, we set its value to be 0.00001, 0.00005, 0.0001, 0.0002, and 0.0005 separately, and fixed the other two hyper-parameter, the critic iteration number and noise length to be 10 and 128 respectively. Then we trained the models for 100 epoches and generated several sequences for a random batch of difficult training folds (For each fold we generated only one sequence, and the batch size was64). Based on these sequences we calculated the critic loss, the nonsense sequence ratio, the padding ratio and the sequence novelty according to the methods mentioned previously. Figure 3.5 shows the results we got. From Figure 3.5-(b), the figure about the nonsense sequence ratio we can see that at the beginning the model can generator a large ratio of the nonsense sequences. However, to check the padding ratio and the sequence novelty we need to generate enough valid sequences, and in the real experiments we found it can be rather hard and time-consuming to satisfy this requirement for the firset several epoches, therefore we only check these two criteria for the last 50 epoches. Also, when the initial learning rate is too large (0.0005 in Figure 3.5-(b)), the nonsense sequence ratio can be always 1 for all the epoches, which means the model can only generate sequences full of padding. Then it is also unfeasible for calculating the next two criteria and we can directly discard this value due to the bad performance, so we only considered the initial learning rate to be 0.00001, 0.00005, 0.0001 and 0.0002 for the padding ratio and the sequence novelty. In Figure 3.5 we can see that the initial learning rate of 0.0001 has the best critic loss and nonsense ratio. The initial learning rate of 0.0001 and 0.0002 have similar performance, but the former has a lower sequence identity. As a result, we chose 0.0001 to be the best initial learning rate and fix it for the next steps of hyper-parameter tuning.

3.3.2 Critic Iteration Number

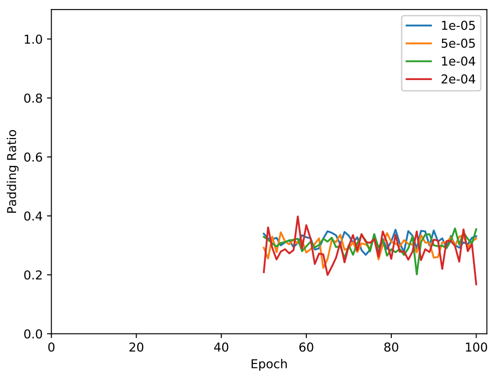
As we discussed in Section 2.6.1, we also modified the critic iteration number in each iteration of the training process. We firstly fixed the learning rate to be 0.0001, which is the best value we got from the previous step, and noise length to be 128, which is the original value in [62]. Then we set the critic iteration number to be 5, 10 and 20 respectively. The results are shown in Figure 3.6. According to the critic loss and nonsense sequence ratio, we got the result that when the number is 20 the model performed the best, for it has the lowest critic loss and the lowest nonsense sequence ratio at the end. For the sequence identity and padding ratio, we can see that they do not vary much as the critic iteration number changes. As a result, we fixed the critic iteration number to be 20 as the best value for this hyper-parameter at the end.



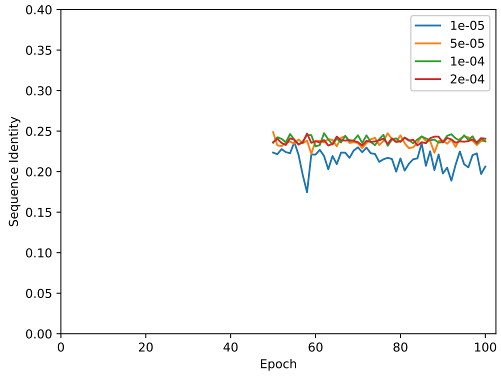
(a) Critic Loss



(b) Nonsense Sequence Ratio



(c) Padding Ratio

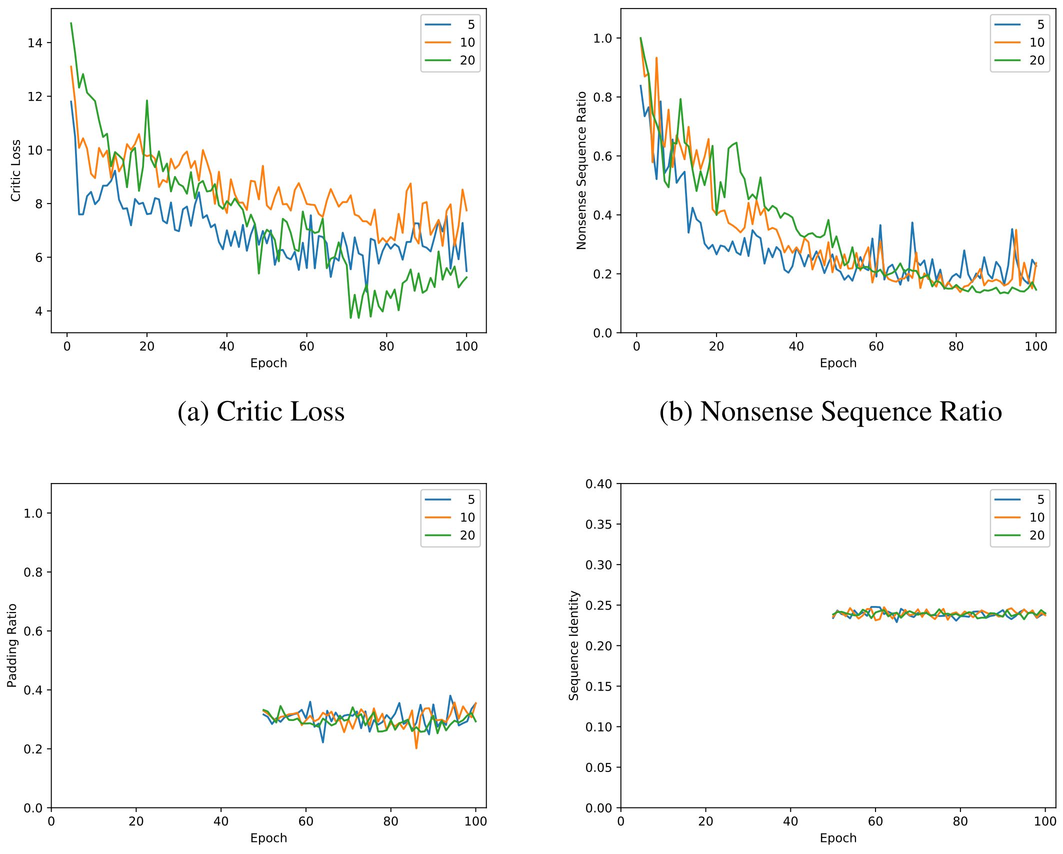


(d) Sequence Identity

Figure 3.5: Comparison for different initial learning rates when critic iteration number is fixed to 10 and noise length is fixed to 128.

3.3.3 Noise Length

The last hyper-parameter for cWGAN is the noise length, which means the dimension of the input noise. In this project we took a high dimensional Gaussian noise to be the random input, which means each entries in the input noise vector is a random variable of normal distribution and they are independent. As we input the generator with the random noise vector and the fold coordinate, it will map the noise distribution to the protein sequence distribution of the related fold. We then fixed the learning rate and critic iteration number to be 0.0001 and 20 respectively and set the noise length to be 64, 128 and 256. After 100 epoches, we finally selected 128 as the best value according to the results shown in Figure 3.7. Similar to the critic iteration number, in Figure 3.7



(a) Critic Loss (b) Nonsense Sequence Ratio

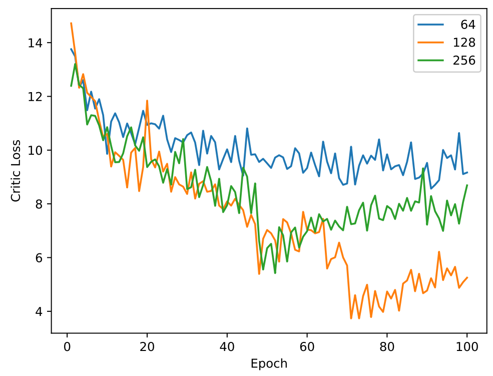
(c) Padding Ratio (d) Sequence Identity

Figure 3.6: Comparison for different critic iteration numbers when initial learning rate is fixed to 0.0001 and noise length is fixed to 128.

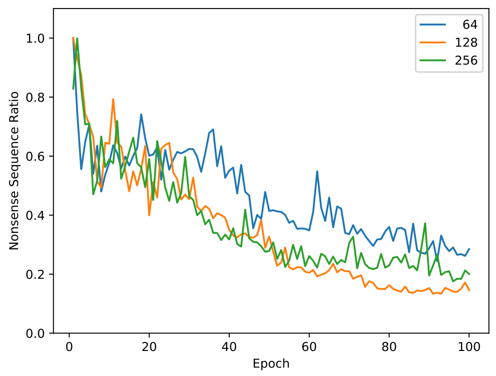
the sequence identity and the padding ratio do not change much for different noise length, but the critic loss and nonsense sequence ratio reach the lowest when we set this hyper-parameter to be 128.

3.3.4 Weight of Feedback Penalty in gcWGAN (λ_2)

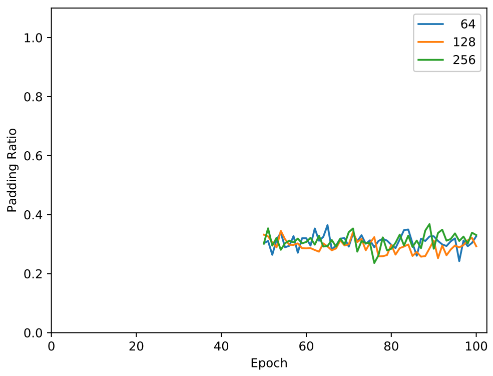
For the gcWGAN model we added a new item, the feedback penalty from the oracle, in the loss function, so there is another hyper-parameter which is the weight of this penalty λ_2. Considering the computational budget we fixed the previous three hyper-parameters to be the best ones we got from cWGAN, and then tried to set this hyper-parameter to be 0.001, 0.01, 0.1, 1 and 10



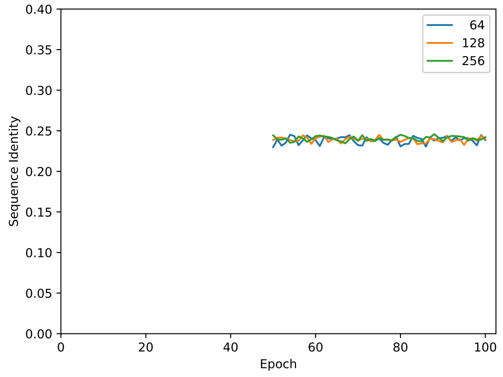
(a) Critic Loss



(b) Nonsense Sequence Ratio



(c) Padding Ratio



(d) Sequence Identity

Figure 3.7: Comparison for different noise length when initial learning rate is fixed to 0.0001 and critic iteration number is fixed to 20.

separately. We also calculated the feedback penalty at the end of each epoch by generating 100 valid sequences for each of the folds in the validation set. According to the results we found that the the model will diverge if λ_2 is larger than or equal to 0.1. Based on the oracle's accuracy on the validation folds, we selected 0.01 as the best value of the weight λ_2 at the end.

3.4 Yield Ratio

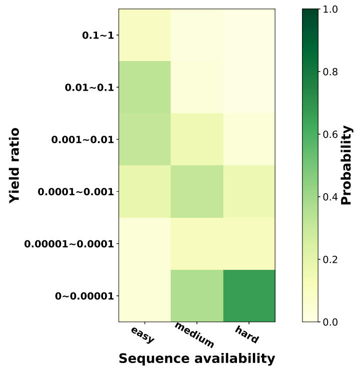
After we have already tuned all the hyper-parameters and the models are well trained, we then need to evaluate the performance of them. The main criteria of the evaluation is the yield ratio, which we defined in Section 2.6.2 as the ratio of successful sequences in all the valid ones generated by the model. In order to tell the difference on yield ratio for different folds and avoid a 0 yield ratio for too many folds, we applied the assess loop mentioned in Section 2.6.2 to calculate the yield ratios for each of the folds in training, validation and test sets, and then took an average. We also calculated the average yield ratio for different fold classes, sequence availability ranks and the folds for which the oracle's accuracy locate in different ranges. The results are shown in Table 3.2, and we can see that generally the performance of the gcWGAN is a little better than that of cWGAN with a feedback from the oracle, even though for some classes and dataset cWGAN may perform better. The yield ratio varies for different classes and sequence availability ranks. As a result, we would also like to discover the factors that my influence this criteria.

Yield Ratio	cWGAN			guided-cWGAN		
	train	validation	test	train	validation	test
all	1.3e-2	6.7e-4	2.1e-3	1.3e-2	9.2e-4	2.4e-3
a	2.0e-3	6.1e-5	1.2e-4	4.2e-3	1.9e-4	3.5e-4
b	2.8e-2	2.6e-3	7.0e-3	2.2e-2	2.0e-3	6.6e-3
c	4.6e-2	1.9e-3	9.8e-3	4.9e-2	3.2e-3	1.3e-2
d	5.2e-3	4.2e-4	7.8e-4	6.0e-3	9.5e-4	1.1e-3
e	1.1e-4	< 1e-5	< 1e-5	1.2e-3	< 1e-5	< 1e-5
f	4.9e-2	2.4e-4	4.1e-5	3.3e-2	4.4e-4	< 1e-5
g	2.1e-4	9.3e-6	2.5e-4	5.6e-4	1.4e-5	3.9e-4
easy	3.5e-2	4.2e-3	2.1e-2	3.8e-2	7.9e-3	2.2e-2
medium	1.3e-2	2.1e-3	1.3e-3	1.1e-2	1.4e-3	1.4e-3
hard	2.0e-3	2.9e-4	3.6e-4	2.2e-3	4.2e-4	6.5e-4
oracle accuracy	train	validation	test	train	validation	test
$0 \sim 0.25$	< 1e-5	1.6e-5	< 1e-5	2.9e-5	1.1e-5	< 1e-5
$0.25 \sim 0.5$	1.1e-3	2.2e-5	< 1e-5	7.8e-4	5.1e-5	2.6e-5
$0.5 \sim 0.75$	2.5e-3	1.6e-3	1.3e-3	4.3e-3	3.1e-3	2.1e-3
$0.75 \sim 1$	5.1e-2	3.2e-3	1.7e-2	4.8e-2	4.6e-3	1.9e-2

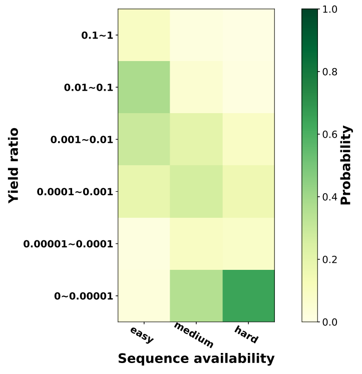
Table 3.2: Yield ratio results for cWGAN and guided-cWGAN (gcWGAN).

3.4.1 Effect of Data Availability on the Yield Ratio

The sequence availability is the number of natural sequences for a fold in our dataset. Empirically, a model can be trained better with a larger dataset, so we think the sequence availability can infulence the yield ratio as it should be higher with a higher sequence availability. We then applied heatmap plots to show the the folds counts in different yield ratio range for different sequence availability ranks. As the sequences in the validation and test sets were not used during the whole process of training, we only took this test for the training set and checked the result both for the cWGAN and gcWGAN models. The heatmap plots are shown in Figure 3.8, while we normalized the data for each columns as we would like to see the yield ratio distribution for different sequence availability ranks. We can see that for folds with high sequence availability (easy case), there can be more folds locating in the range of high yield ratio, while for the hard case most folds are related to yield ratios in the lowest range. The results for the two models are similar. Then we can get a conclusion that the sequence availability did effect the yield ratio, as the folds with more natural sequences for training are more likely to get a high yield ratio.



(a) cWGAN

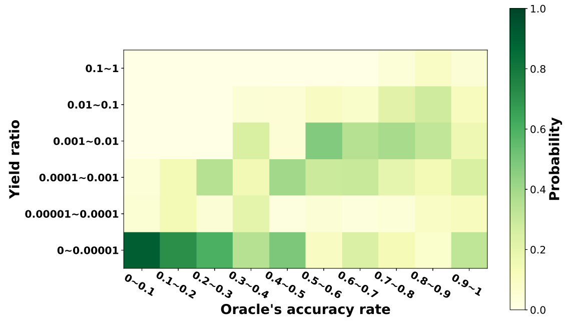


(b) gcWGAN

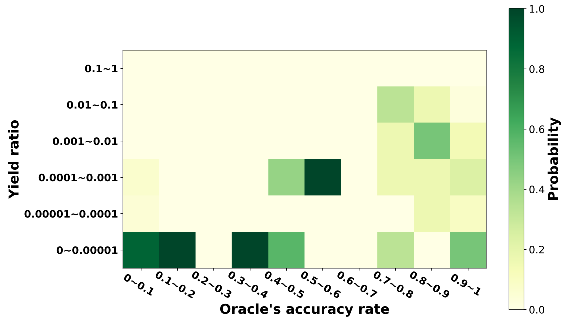
Figure 3.8: Effect of sequence availability on yield ratio for training, validation and test sets for cWGAN and gcWGAN.

3.4.2 Effect of Oracle's Accuracy on the Yield Ratio

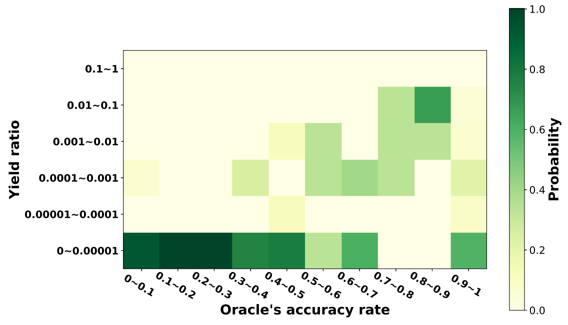
For the gcWGAN model we took the feedback from the oracle as an item in the loss function, so the performance, or in other words, the oracle's accuracy may also influence the performance of the gcWGAN model. Also, as the oracle to evaluate the final performance of the model, its accuracy will have a great effect on the evalustion. From Table 3.2 we can see that as the oracle's accuracy increases, the average yield ratio also rises, and this result holds for all of the training, validation and test sets. We also applied heatmap plots to show the the folds counts in different range of yield ratios for different oracle accuracy range based on the gcWGAN model. We did this for the training, validation and test set separatly, and the results are shown in Figure 3.9, and as the oracle's accuracy increases, there can be more folds locating in the high yield ratio ranges.



(a) Training set



(b) Validation set



(c) Test set

Figure 3.9: Effect of Oracle's (DeepSF) error on yield ratio for training, validation and test sets for gcwGAN.

3.5 Frechet Protein Distance (FPD)

According to previous researches it can be notoriously hard to assess the performance of GANs, and GANs are very likely to have mode collapse. Motivated from Frechet Inception Distance (FID) score for image generation [63] and Frechet ChemNet Distance (FCD) for chemical generation [64], we developed an Frechet Protein Distance (FPD) to evaluate whether our gcWGAN is also suffering from model collapse or not. Similarly to FID and FCD, FPD is a measure of distance with the assumption Gaussian distribution for a given mean and covariance matrix (since Gaussian distribution is the only distribution that maximize the entropy for a given covariance and mean.). FPD is used to measure the distance between generated sequences and the real ones, as the lower the FPD score, the more similar they are. We Modified DeepSF model to predict 7 fold classes based on one-hot encoding sequences only. We added another fully connected with 50 neurons before the softmax layer. We trained the model on the same training/test sets we did for the oracle and we achieved a 95% and 77% accuracy for training and test respectively. We consider the new fully connected layer as the final feature representation for fold classes. We generated same number of sequences for each fold classes as we have in nature. Then, we calculate the pairwise FPD between for pair of fold class between generated ones and the real one. The result for the FPD calculation is shown in Figure 3.10. Based on the results, diagonal elements have lower one in comparison to the off diagonal. It statistically shows that gcWGAN did not suffer from mode collapse. We then performed one-sided paired t-test between each diagonal element and the off-diagonal in the same column based on Figure 3.10. We found out that the p-value is equal to 0.03 which we can conclude that in average diagonal elements are statistically lower than the off-diagonal in the same column.

3.6 Case Studies

Since we have hundreds of folds used in our dataset and it can be rather resource-consuming to make a deeper analysis for all of them, we selected 6 representative folds to reflect the performance

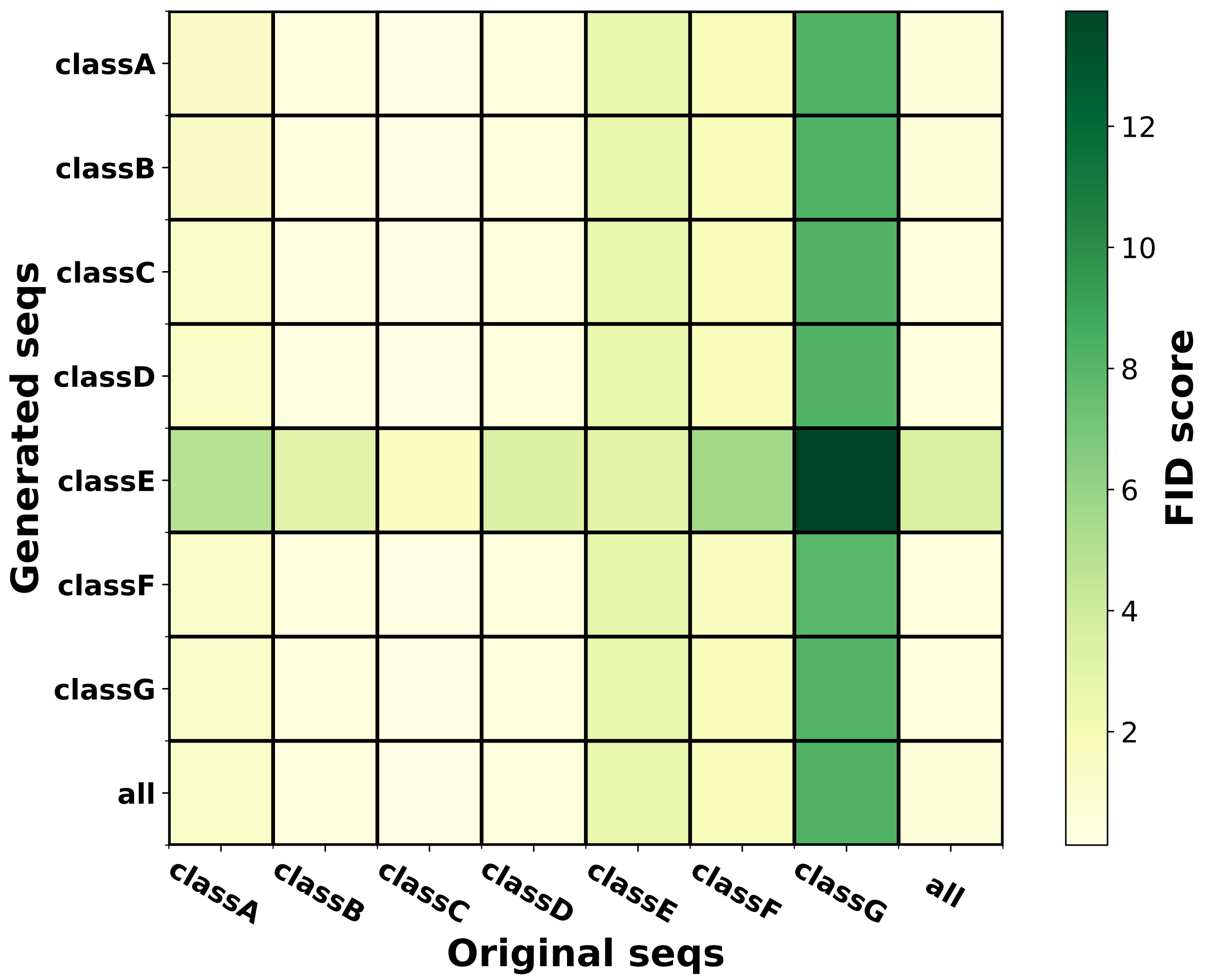


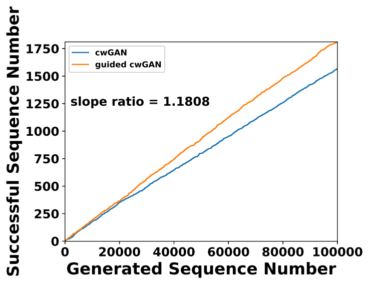
Figure 3.10: Comparing generated sequences from gcWGAN with the original sequences with FID score.

for different cases. The folds were selected from the three sequence availability ranks separately, and for each case one fold is of a high yield ratio (higher than 0.0001) and the other is of a low yield ratio (lower than 0.0001). The folds selected are shown in Table 3.3.

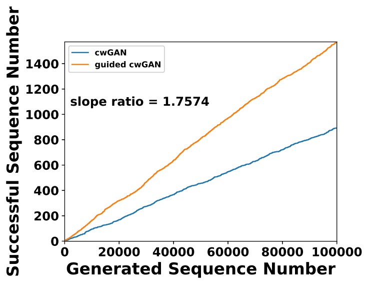
yield ratio	easy	medium	hard
high	b.2	c.94	c.56
low	a.35	g.44	d.107

Table 3.3: Case studies fold from test set to represent the whole set.

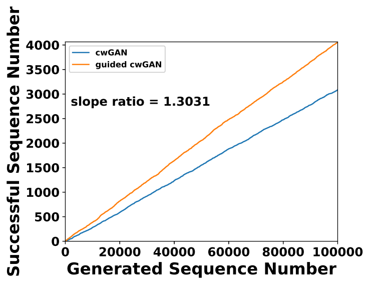
For each of the case we generated 100,000 valid sequences and recorded the index of the successful ones, and then we can calculate the successfully generating rate to see how fast we can get successful ones as we keep generating sequences for these 6 cases. From Figure 3.11 we can see that it is a linear relationship between the successful sequence number and the number of generated ones, and the slope can reflect the successfully generating rate. For all of the 6 cases, the gcWGAN model performs better than cWGAN as it has a higher slope. Especially for the fold d.107, we could not get even one successful one by generating 100,000 sequences based on cWGAN but there was a successful one based on gcWGAN. We also calculated the slope ratio to see the improvement from cWGAN to gcWGAN, and it varies for different folds. For folds with low yield ratios, we are more likely to get a larger improvement. In summary, the gcWGAN can get successful sequences faster than cWGAN.



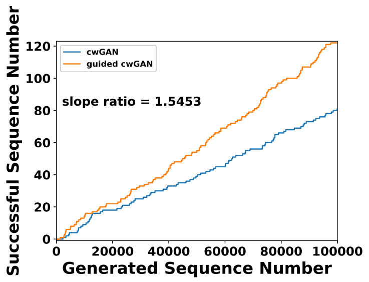
(a) b.2



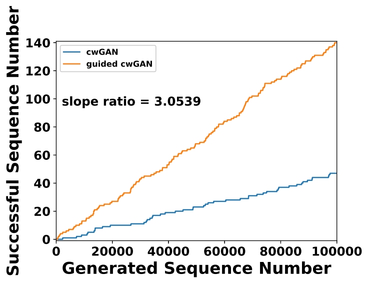
(b) c.94



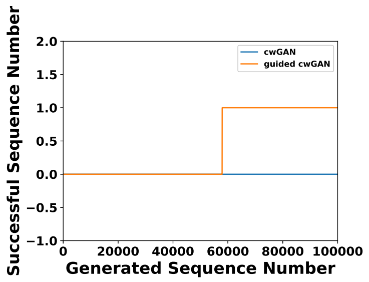
(c) c.56



(d) a.35



(e) g.44

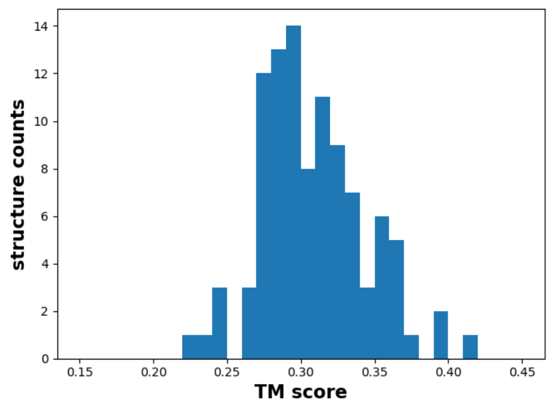


(f) d.107

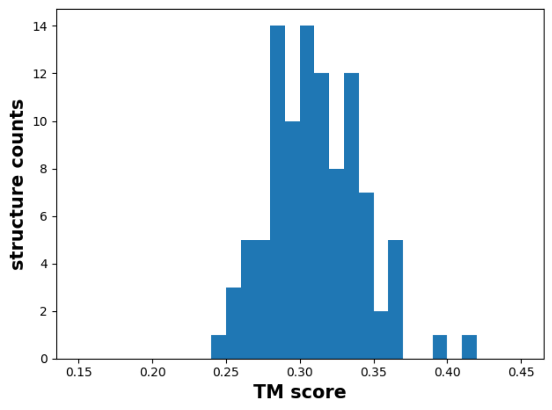
Figure 3.11: Successful sequence number versus generated sequence number for selected folds.

3.7 Performance on a Novel Fold

Besides the 1,232 folds we got from SCOPe, we also introduced a novel fold that has never been seen in all the previous steps. Therefore we can test the whole system based on it, including the fold representation, the generating model and our oracle. We firstly generated 1,000 successful sequences for this novel fold, and then pursued the funneling filters processes mentioned in Section 2.7, and we finally selected 10 sequences sent to the Rosetta software. In Rosetta, we got 10,000 structure predictions for each sequence and then we calculate their coordinates according to our fold representation method. For each sequence we selected the top ten percent (1,000 sequences) with the lowest $l2$ norm (the $l2$ distance between the coordinates of generated sequences and that of the ground truth). For the 1,000 sequences we then calculated the root-mean-square deviation of atomic positions (or simply root-mean-square deviation, RMSD) between each pair of them and got an 1000×1000 RMSD matrix. After that we tried two clustering methods separately for further funneling process. Firstly we applied spectral clustering algorithm [65] with the RMSD matrix to get 10 clusters and for each cluster we chose the structure with the lowest $l2$ distance to the ground truth. As a result we got 100 structure predictions for the generated sequences based on the gcWGAN model. We also applied k-means algorithm in the 20 dimensional coordinate space and got 10 clusters as well. By going through the same selecting process we then got another 100 structure predictions. For these predictions we calculated the TM scores between them and the real structure of the novel fold. In Figure 3.12 we show the distribution of the TM scores both for the two methods. We can see that both two distributions locate in the range around 0.3 and the minimum TM score is larger than 0.2. We had generated some sequences whose structure predictions can be very close to the real one as the TM scores between them are larger than 0.35, which means our system is effective on a novel fold that has never been seen before.



(a) Structures selected based on spectral clustering algorithm



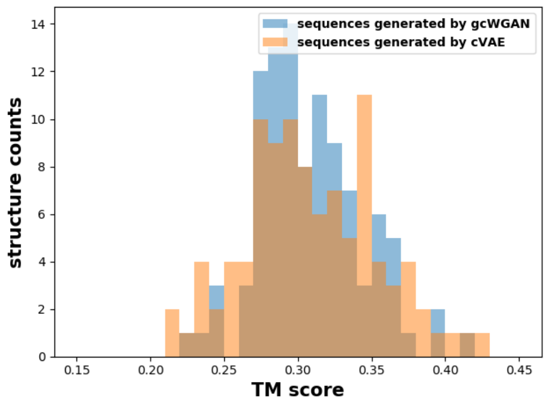
(b) Structures selected based on k-means clustering algorithm

Figure 3.12: TM score distribution of the selected structure predictions for the novel fold on gcWGAN.

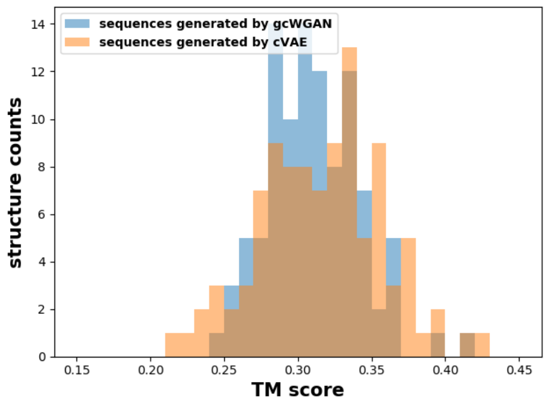
3.8 Model Performance Compared to the cVAE Model

In [66] they also provided a conditional variational autoencoder (cVAE) model that can generate protein sequences taking different folds as the conditions. We generated 1,000 sequences for the novel fold based on their model and selected 10 according to the same process as we did in Section 2.7. We sent them to the Rosetta software to get 10,000 structure predictions for each. We then applied the same two selecting methods in the previous section based on the $l2$ distances of the coordinates and the clustering algorithms, and also chose two sets of predictions with a size of 100. Figure 3.13 shows the TM score distribution for these two sets, and we also plotted that of our gcWGAN model for comparison. We can see both two models have a similar result, which means our model has a similar accuracy performance to the cVAE model.

We then compared the performance between their model and ours by calculating pairwise sequence identities to show the sequence diversity and novelty, as our goal is to to design some novel folds that has not ever seen in the nature. We did this comparison in the previously mentioned six cases in Section 3.6. For the six chosen folds, we separately selected 100 sequences from the generated ones (sequence without passing the DeepSF filter), 100 from the successful ones (sequences



(a) Structures selected based on spectral clustering algorithm

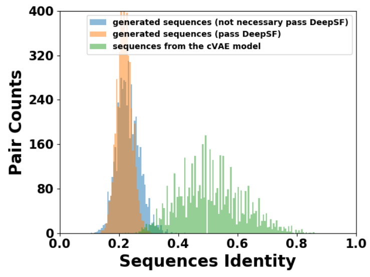


(b) Structures selected based on k-means clustering algorithm

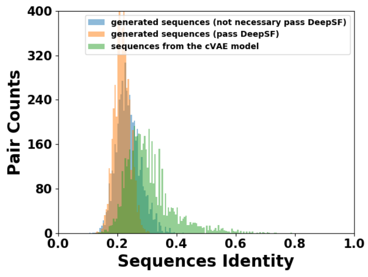
Figure 3.13: TM score distribution of the selected structure predictions for the novel fold on gcWGAN and cVAE.

that passed the DeepSF filter) and 100 generated by the cVAE model to pursue the following analysis. To find out the sequence diversity, we calculated the pairwise sequence identities for all the pairs in the 100 sequences and plotted the hist graphs as shown in Figure 3.14 and Figure 3.15, while Figure 3.14 shows the results for the cWGAN model and Figure 3.15 shows that for the gcWGAN model. From the results we can see that the pairwise sequence identities distribution is more concentrated and locates more close to a low value range for our models, which means we can generate sequences with a higher diversity. Therefore, our model can discover a larger space for generated sequences to be folding into the target folds.

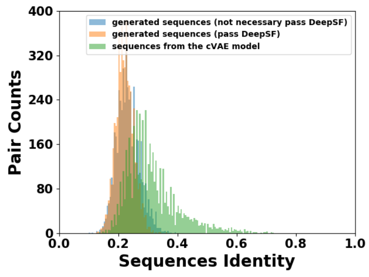
To reflect the sequence novelty, we calculated the sequence identities between the generated ones and the natural sequences related to the same target fold for each selected sequence. Then among all the values we got for a certain sequence, we selected the maximum one which reflect the similarity between the sequence and its closest natural one. We plotted the hist graphs for these sequences as shown in Figure 3.16 and Figure 3.17, while Figure 3.16 shows the results for the cWGAN model and Figure 3.17 shows that for the gcWGAN model. Similar to the sequence diversity, the closer the distribution to the low value, the higher novelty it reveals. Then from the results we can see that the sequence novelty for our model is a little better or at least no worse than



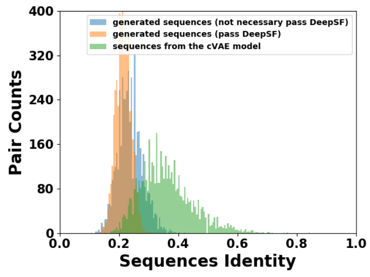
(a) b.2



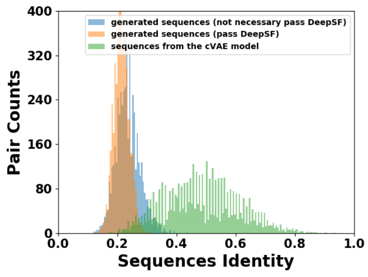
(b) c.94



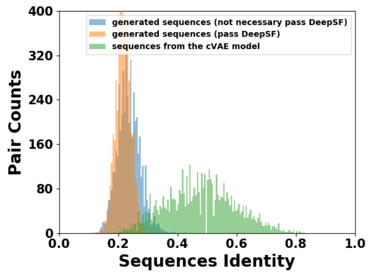
(c) c.56



(d) a.35

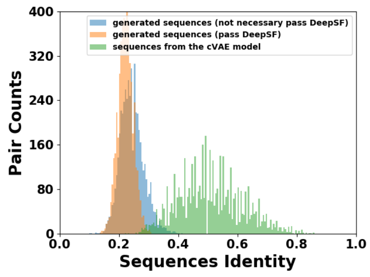


(e) g.44

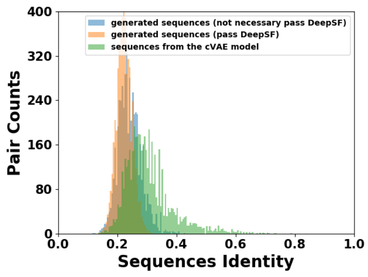


(f) d.107

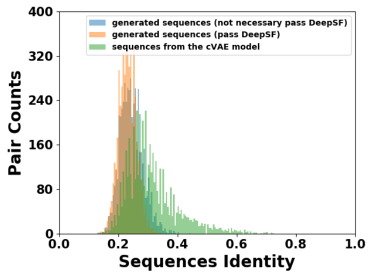
Figure 3.14: Sequence identity distribution based on cwGAN.



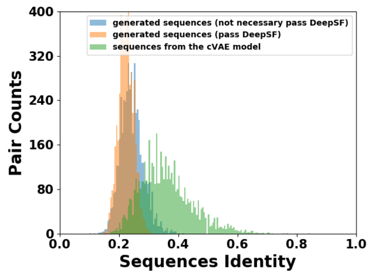
(a) b.2



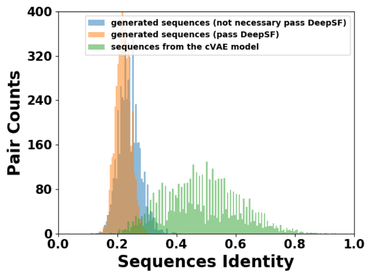
(b) c.94



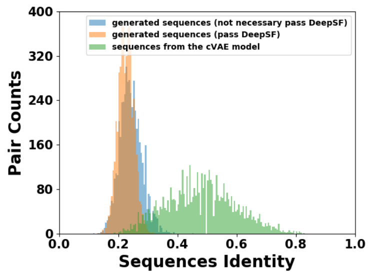
(c) c.56



(d) a.35



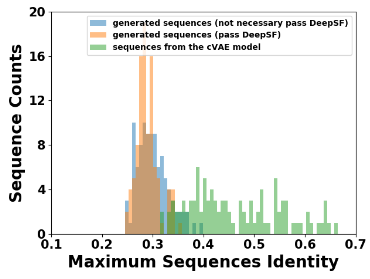
(e) g.44



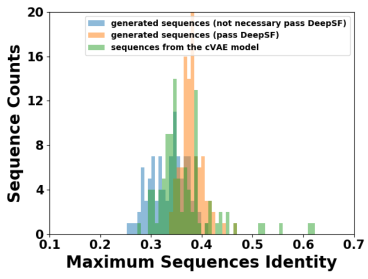
(f) d.107

Figure 3.15: Sequence identity distribution based on gcwGAN.

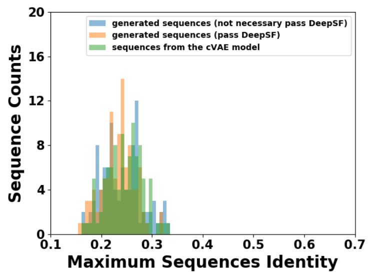
that of the cVAE model, Especially for the fold b.2 our models have made a great improvement on the sequence novelty. As a result, our models are more likely to generate some novel folds.



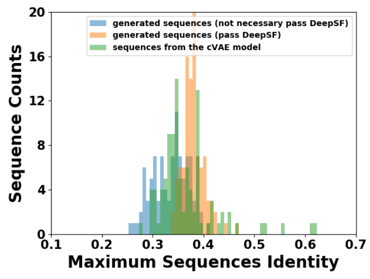
(a) b.2



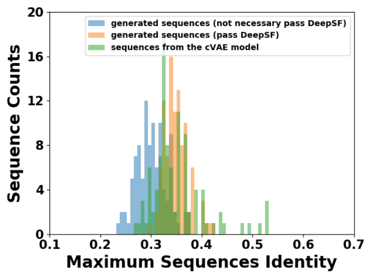
(b) c.94



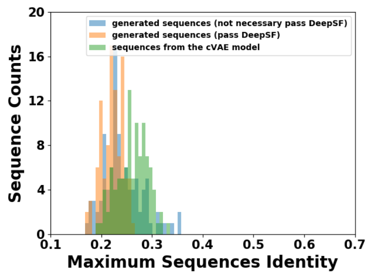
(c) c.56



(d) a.35

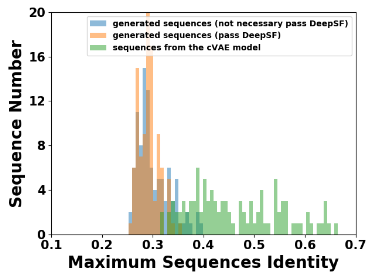


(e) g.44

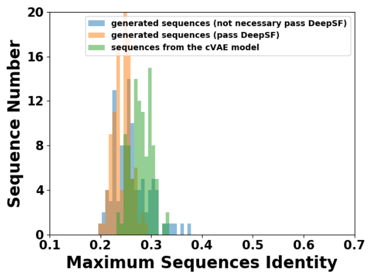


(f) d.107

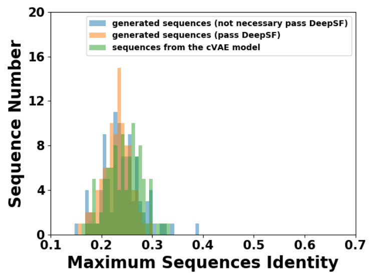
Figure 3.16: Sequence novelty distribution based on cwGAN.



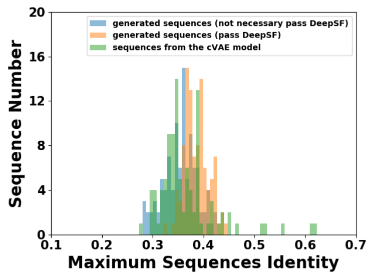
(a) b.2



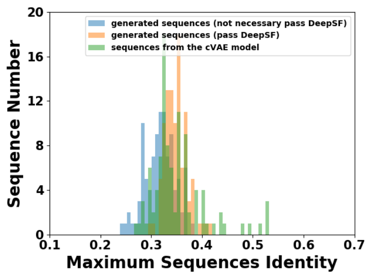
(b) c.94



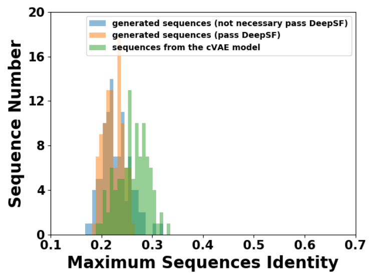
(c) c.56



(d) a.35



(e) g.44



(f) d.107

Figure 3.17: Sequence novelty distribution based on gcWGAN.

4. SUMMARY AND CONCLUSIONS

In this project we mainly designed two novel deep generating models for *de novo* protein sequence design on desired protein folds. During the training process, we both utilized the sequence data with and without structure information in a semi-supervised process. We also took a fast oracle model which can get fold predictions and provide a feedback for the training steps, so that the whole process formed a closed loop. According to the funneling filters we designed and the final results, our models can generate and incrementally select protein sequences for desired folds. We studied the sequence generation for nearly 800 folds that belong to different classes and different sequence availability range, where over 100 folds of the 800 are not seen in the training process so that they can be regarded as novel folds for the model. As we comprehensively assessed the performances through sequence and structure analysis, our gcWGAN model finally reached a average yield ratio of 0.24% for all the test folds, and a much higher yield ratio of 2.2% for the easy folds in the test set. Then we can get a conclusion that our models can generate protein sequences targeting at desired folds with a high rate, which reveals the value of current protein data for unraveling sequence-structure relationship and utilizing resulting knowledge for inverse protein design. And with a enlarged dataset for each fold we are likely to get a better performance, since the performance for the folds with a larger sequence availability is better.

Compared to another sequence generating model, the conditional variational auto-encoder (cVAE), recently used for novel fold generation [66], the WGAN with its implicit model and Wasserstein distance can overcome some known limitations of VAE such that it will get an approximation of the posterior mostly with Gaussian distributions, which can lead to underestimation of the variance [67], and its mean field assumption ignores the dependencies between different factorization components in their latent space, which could significantly reduce their representation power [67, 68]. Moreover, for VAE, using of KL-divergence does not have low dimensional support in high dimensional problems [61], and the VAE's zero-centering assumption in prior can lead to poor inference and generation [69]. According to the results, our gcWGAN can finally reach a similar accuracy to the cVAE model with a higher sequence diversity and novelty.

REFERENCES

[1] B. Alberts, D. Bray, K. Hopkin, A. D. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Walter, *Essential Cell Biology*. Garland Science, 2015.

[2] D. Baker and A. Sali, "Protein structure prediction and structural genomics," *Science*, vol. 294, no. 5540, pp. 93–96, 2001.

[3] C. Pabo, "Molecular technology: designing proteins and peptides," *Nature*, vol. 301, no. 5897, pp. 200–200, 1983.

[4] A. G. Street and S. L. Mayo, "Computational protein design," *Structure*, vol. 7, no. 5, pp. R105–R109, 1999.

[5] C. B. Anfinsen, M. Sela, and J. P. Cooke, "The reversible reduction of disulfide bonds in polyalanyl ribonuclease," *Journal of Biological Chemistry*, vol. 237, no. 6, pp. 1825–1831, 1962.

[6] C. B. Anfinsen, "Principles that govern the folding of protein chains," *Science*, vol. 181, no. 4096, pp. 223–230, 1973.

[7] R. Sheridan, R. J. Fieldhouse, S. Hayat, Y. Sun, Y. Antipin, L. Yang, T. Hopf, D. S. Marks, and C. Sander, "Evfold. org: Evolutionary couplings and protein 3d structure prediction," *bioRxiv*, p. 021022, 2015.

[8] M. Weigt, R. A. White, H. Szurmant, J. A. Hoch, and T. Hwa, "Identification of direct residue contacts in protein–protein interaction by message passing," *Proceedings of the National Academy of Sciences*, vol. 106, no. 1, pp. 67–72, 2009.

[9] H. Kamisetty, S. Ovchinnikov, and D. Baker, "Assessing the utility of coevolution-based residue–residue contact predictions in a sequence-and structure-rich era," *Proceedings of the National Academy of Sciences*, p. 201314045, 2013.

[10] S. Seemayer, M. Gruber, and J. Söding, “CcmpredâĂŤfast and precise prediction of protein residue–residue contacts from correlated mutations,” *Bioinformatics*, vol. 30, no. 21, pp. 3128–3130, 2014.

[11] T. A. Hopf, C. P. Schärfe, J. P. Rodrigues, A. G. Green, O. Kohlbacher, C. Sander, A. M. Bonvin, and D. S. Marks, “Sequence co-evolution gives 3d contacts and structures of protein complexes,” *Elife*, vol. 3, p. e03430, 2014.

[12] D. E. Kim, F. DiMaio, R. Yu-Ruei Wang, Y. Song, and D. Baker, “One contact for every twelve residues allows robust and accurate topology-level protein structure modeling,” *Proteins: Structure, Function, and Bioinformatics*, vol. 82, pp. 208–218, 2014.

[13] S. Ovchinnikov, H. Kamisetty, and D. Baker, “Robust and accurate prediction of residue–residue interactions across protein interfaces using evolutionary information,” *Elife*, vol. 3, p. e02030, 2014.

[14] S. Wang, S. Sun, Z. Li, R. Zhang, and J. Xu, “Accurate de novo prediction of protein contact map by ultra-deep learning model,” *PLoS Computational Biology*, vol. 13, no. 1, p. e1005324, 2017.

[15] J. Zhu, S. Wang, D. Bu, and J. Xu, “Protein threading using residue co-variation and deep learning,” *Bioinformatics*, vol. 34, no. 13, pp. i263–i273, 2018.

[16] N. Koga, R. Tatsumi-Koga, G. Liu, R. Xiao, T. B. Acton, G. T. Montelione, and D. Baker, “Principles for designing ideal protein structures,” *Nature*, vol. 491, no. 7423, p. 222, 2012.

[17] P. Gainza, H. M. Nisonoff, and B. R. Donald, “Algorithms for protein design,” *Current Opinion in Structural Biology*, vol. 39, pp. 16–26, 2016.

[18] N. A. Pierce and E. Winfree, “Protein design is np-hard,” *Protein Engineering*, vol. 15, no. 10, pp. 779–782, 2002.

[19] P. Gainza, K. E. Roberts, I. Georgiev, R. H. Lilien, D. A. Keedy, C.-Y. Chen, F. Reza, A. C. Anderson, D. C. Richardson, J. S. Richardson, *et al.*, “Osprey: protein design with ensem-

bles, flexibility, and provable algorithms," in *Methods in Enzymology*, vol. 523, pp. 87–107, Elsevier, 2013.

[20] S. Traoré, D. Allouche, I. André, S. De Givry, G. Katsirelos, T. Schiex, and S. Barbe, "A new framework for computational protein design through cost function network optimization," *Bioinformatics*, vol. 29, no. 17, pp. 2129–2136, 2013.

[21] M. A. Hallen and B. R. Donald, "Comets (constrained optimization of multistate energies by tree search): A provable and efficient protein design algorithm to optimize binding affinity and specificity with respect to sequence," *Journal of Computational Biology*, vol. 23, no. 5, pp. 311–321, 2016.

[22] M. Karimi and Y. Shen, "icfn: an efficient exact algorithm for multistate protein design," *Bioinformatics*, vol. 34, no. 17, pp. i811–i820, 2018.

[23] C. L. Kingsford, B. Chazelle, and M. Singh, "Solving and analyzing side-chain positioning problems using linear and integer programming," *Bioinformatics*, vol. 21, no. 7, pp. 1028–1039, 2004.

[24] M. Fromer and C. Yanover, "A computational framework to empower probabilistic protein design," *Bioinformatics*, vol. 24, no. 13, pp. i214–i222, 2008.

[25] D. T. Jones, "De novo protein design using pairwise potentials and a genetic algorithm," *Protein Science*, vol. 3, no. 4, pp. 567–574, 1994.

[26] A. Leaver-Fay, M. Tyka, S. M. Lewis, O. F. Lange, J. Thompson, R. Jacak, K. W. Kaufman, P. D. Renfrew, C. A. Smith, W. Sheffler, *et al.*, "Rosetta3: an object-oriented software suite for the simulation and design of macromolecules," in *Methods in Enzymology*, vol. 487, pp. 545–574, Elsevier, 2011.

[27] T. Kortemme, L. A. Joachimiak, A. N. Bullock, A. D. Schuler, B. L. Stoddard, and D. Baker, "Computational redesign of protein-protein interaction specificity," *Nature Structural and Molecular Biology*, vol. 11, no. 4, p. 371, 2004.

[28] G. Grigoryan, A. W. Reinke, and A. E. Keating, "Design of protein-interaction specificity gives selective bzip-binding peptides," *Nature*, vol. 458, no. 7240, p. 859, 2009.

[29] R. S. Rudicell, Y. Do Kwon, S.-Y. Ko, A. Pegu, M. K. Louder, I. S. Georgiev, X. Wu, J. Zhu, J. C. Boyington, X. Chen, *et al.*, "Enhanced potency of a broadly neutralizing hiv-1 antibody in vitro improves protection against lentiviral infection in vivo," *Journal of Virology*, pp. JVI–02213, 2014.

[30] P.-S. Huang, S. E. Boyken, and D. Baker, "The coming of age of de novo protein design," *Nature*, vol. 537, no. 7620, p. 320, 2016.

[31] S. Chaudhury, S. Lyskov, and J. J. Gray, "Pyrosetta: a script-based interface for implementing molecular modeling algorithms using rosetta," *Bioinformatics*, vol. 26, no. 5, pp. 689–691, 2010.

[32] S. J. Fleishman, A. Leaver-Fay, J. E. Corn, E.-M. Strauch, S. D. Khare, N. Koga, J. Ashworth, P. Murphy, F. Richter, G. Lemmon, *et al.*, "Rosettascripts: a scripting language interface to the rosetta macromolecular modeling suite," *PloS One*, vol. 6, no. 6, p. e20161, 2011.

[33] E. Marcos, B. Basanta, T. M. Chidyausiku, Y. Tang, G. Oberdorfer, G. Liu, G. Swapna, R. Guan, D.-A. Silva, J. Dou, *et al.*, "Principles for designing proteins with cavities formed by curved β sheets," *Science*, vol. 355, no. 6321, pp. 201–206, 2017.

[34] H. Shen, J. A. Fallas, E. Lynch, W. Sheffler, B. Parry, N. Jannetty, J. Decarreau, M. Wagenbach, J. J. Vicente, J. Chen, *et al.*, "De novo design of self-assembling helical protein filaments," *Science*, vol. 362, no. 6415, pp. 705–709, 2018.

[35] J. Dou, A. A. Vorobieva, W. Sheffler, L. A. Doyle, H. Park, M. J. Bick, B. Mao, G. W. Foight, M. Y. Lee, L. A. Gagnon, *et al.*, "De novo design of a fluorescence-activating β-barrel," *Nature*, vol. 561, no. 7724, p. 485, 2018.

[36] D.-A. Silva, S. Yu, U. Y. Ulge, J. B. Spangler, K. M. Jude, C. Labão-Almeida, L. R. Ali, A. Quijano-Rubio, M. Ruterbusch, I. Leung, *et al.*, "De novo design of potent and selective mimics of il-2 and il-15," *Nature*, vol. 565, no. 7738, p. 186, 2019.

[37] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” in *Advances in Neural Information Processing Systems*, pp. 2672–2680, 2014.

[38] D. P. Kingma and M. Welling, “Auto-encoding variational bayes,” *arXiv preprint arXiv:1312.6114*, 2013.

[39] A. Gupta and J. Zou, “Feedback gan (fbgan) for dna: a novel feedback-loop architecture for optimizing protein functions,” *arXiv preprint arXiv:1804.01694*, 2018.

[40] N. Killoran, L. J. Lee, A. Delong, D. Duvenaud, and B. J. Frey, “Generating and designing dna with deep generative models,” *arXiv preprint arXiv:1712.06148*, 2017.

[41] P. Eastman, J. Shi, B. Ramsundar, and V. S. Pande, “Solving the RNA design problem with reinforcement learning,” *PLoS Computational Biology*, vol. 14, p. e1006176, June 2018.

[42] M. Popova, O. Isayev, and A. Tropsha, “Deep reinforcement learning for de novo drug design,” *Science Advances*, vol. 4, p. eaap7885, July 2018.

[43] G. L. Guimaraes, B. Sanchez-Lengeling, C. Outeiral, P. L. C. Farias, and A. Aspuru-Guzik, “Objective-Reinforced Generative Adversarial Networks (ORGAN) for Sequence Generation Models,” *arXiv:1705.10843 [cs, stat]*, May 2017. arXiv: 1705.10843.

[44] A. T. Muller, J. A. Hiss, and G. Schneider, “Recurrent neural network model for constructive peptide design,” *Journal of Chemical Information and Modeling*, vol. 58, no. 2, pp. 472–479, 2018.

[45] R. Kolodny, L. Pereyaslavets, A. O. Samson, and M. Levitt, “On the universe of protein folds,” *Annual Review of Biophysics*, vol. 42, pp. 559–582, 2013.

[46] J. Gu and P. E. Bourne, *Structural Bioinformatics*, vol. 44. John Wiley & Sons, 2009.

[47] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, *et al.*, “A density-based algorithm for discovering clusters in large spatial databases with noise.,” in *Kdd*, vol. 96, pp. 226–231, 1996.

[48] L. Jaroszewski, Z. Li, S. S. Krishna, C. Bakolitsa, J. Wooley, A. M. Deacon, I. A. Wilson, and A. Godzik, "Exploration of uncharted regions of the protein universe," *PLoS Biology*, vol. 7, no. 9, p. e1000205, 2009.

[49] B. Schölkopf, A. Smola, and K.-R. Müller, "Kernel principal component analysis," in *International Conference on Artificial Neural Networks*, pp. 583–588, Springer, 1997.

[50] Y. Zhang and J. Skolnick, "Scoring function for automated assessment of protein structure template quality," *Proteins: Structure, Function, and Bioinformatics*, vol. 57, no. 4, pp. 702–710, 2004.

[51] J. Hou, B. Adhikari, and J. Cheng, "Deepsf: deep convolutional neural network for mapping protein sequences to folds," *Bioinformatics*, vol. 34, no. 8, pp. 1295–1303, 2017.

[52] T. Karras, T. Aila, S. Laine, and J. Lehtinen, "Progressive growing of gans for improved quality, stability, and variation," *arXiv preprint arXiv:1710.10196*, 2017.

[53] C. Chu, A. Zhmoginov, and M. Sandler, "Cyclegan: a master of steganography," *arXiv preprint arXiv:1712.02950*, 2017.

[54] H. Zhang, T. Xu, H. Li, S. Zhang, X. Huang, X. Wang, and D. Metaxas, "Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks," *arXiv preprint*, 2017.

[55] Y. Jin, J. Zhang, M. Li, Y. Tian, H. Zhu, and Z. Fang, "Towards the automatic anime characters creation with generative adversarial networks," *arXiv preprint arXiv:1708.05509*, 2017.

[56] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros, "Context encoders: Feature learning by inpainting," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2536–2544, 2016.

[57] G. Antipov, M. Baccouche, and J.-L. Dugelay, "Face aging with conditional generative adversarial networks," in *2017 IEEE International Conference on Image Processing (ICIP)*, pp. 2089–2093, IEEE, 2017.

[58] L.-C. Yang, S.-Y. Chou, and Y.-H. Yang, “Midinet: A convolutional generative adversarial network for symbolic-domain music generation,” *arXiv preprint arXiv:1703.10847*, 2017.

[59] M. Mirza and S. Osindero, “Conditional generative adversarial nets,” *arXiv preprint arXiv:1411.1784*, 2014.

[60] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, “Improved techniques for training gans,” in *Advances in Neural Information Processing Systems*, pp. 2234–2242, 2016.

[61] M. Arjovsky, S. Chintala, and L. Bottou, “Wasserstein generative adversarial networks,” in *International Conference on Machine Learning*, pp. 214–223, 2017.

[62] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, “Improved training of wasserstein gans,” in *Advances in Neural Information Processing Systems*, pp. 5767–5777, 2017.

[63] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, “Gans trained by a two time-scale update rule converge to a local nash equilibrium,” in *Advances in Neural Information Processing Systems*, pp. 6626–6637, 2017.

[64] K. Preuer, P. Renz, T. Unterthiner, S. Hochreiter, and G. Klambauer, “Fréchet chemnet distance: a metric for generative models for molecules in drug discovery,” *Journal of Chemical Information and Modeling*, vol. 58, no. 9, pp. 1736–1741, 2018.

[65] U. Von Luxburg, “A tutorial on spectral clustering,” *Statistics and Computing*, vol. 17, no. 4, pp. 395–416, 2007.

[66] J. G. Greener, L. Moffat, and D. T. Jones, “Design of metalloproteins and novel protein folds using variational autoencoders,” *Scientific Reports*, vol. 8, no. 1, p. 16189, 2018.

[67] M. Yin and M. Zhou, “Semi-implicit variational inference,” *arXiv preprint arXiv:1805.11183*, 2018.

[68] M. D. Hoffman, D. M. Blei, C. Wang, and J. Paisley, “Stochastic variational inference,” *The Journal of Machine Learning Research*, vol. 14, no. 1, pp. 1303–1347, 2013.

[69] T. R. Davidson, L. Falorsi, N. De Cao, T. Kipf, and J. M. Tomczak, “Hyperspherical variational auto-encoders,” *arXiv preprint arXiv:1804.00891*, 2018.